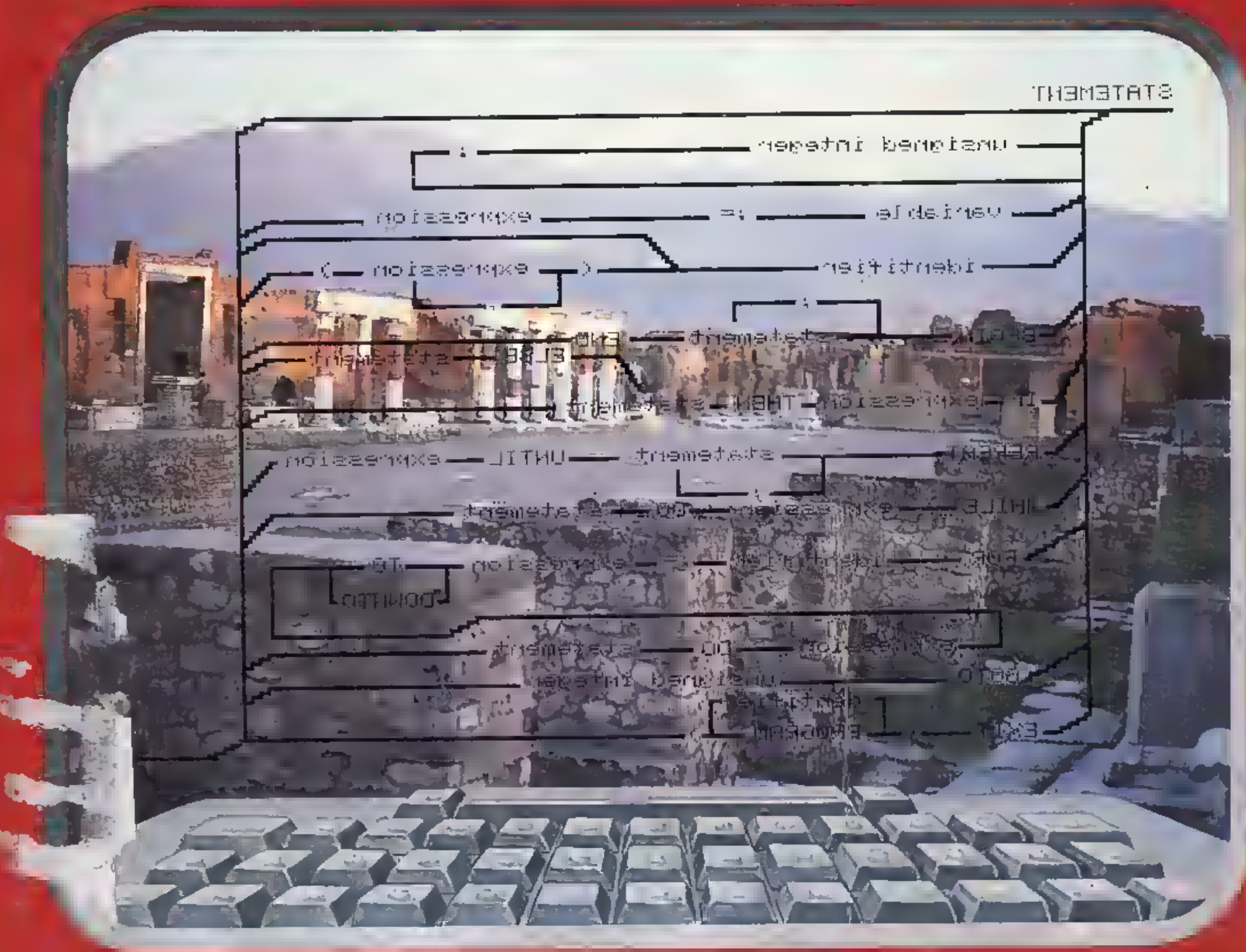
NO. 44 $2.50 JANUARY 1982

MICRO

THE 6502/6809 JOURNAL



Pascal Feature

Experimenters and the Color Computer

RELOC for the Apple

Atari 800 Player/Missile Graphics

TASC™. The Applesoft* Compiler. It turns your Apple into a power tool.

Step up to speed. TASC, the Applesoft Compiler, converts a standard Applesoft BASIC program into super-fast machine code. By increasing program execution speed up to 20 times, Microsoft gives you a power tool for Applesoft BASIC programming.

Highest capacity available.* TASC will compile and run larger programs than any other Applesoft Compiler. As a disk-based system, it doesn't require the simultaneous presence of compiler and program in memory. The memory you save allows you to compile significantly bigger programs.

Power without bulk. Code expansion of up to 100% severely restricts other compilers. TASC's special code compression schemes typically limit code expansion to only 25%. You'll really appreciate that with complex programs or programs that utilize Apple's hi-res graphic pages.

More BASIC power. TASC's powerful new commands increase Applesoft BASIC programming capability. Chain with COMMON allows compiled programs to share variables, so a main menu supports several programs in a single runtime environment.

TASC's True Integer Arithmetic and Integer FOR...NEXT capabilities maximize the execution speed of compiled programs. TASC's near total compatibility with Applesoft speeds compilation of existing programs with little or no modification.

What about mistakes? You perfect your programs interactively with Applesoft. If something does slip by, TASC recovers from errors discovered in compilation and traps all runtime errors. It even permits graceful interruptions during compilation.

See for yourself. Ask for a demonstration of TASC at your Microsoft dealer. Discover the software package that turns your Apple into a power tool.

MICROSOFT
CONSUMER PRODUCTS

A Division of Microsoft Inc.
10700 Northup Way • Bellevue, WA 98004

*Applesoft is a trademark of Apple Computer, Inc.

DATA CAPTURE 4.0©

The most advanced and easiest to use telecommunications program for use with the MICROMODEM II™ or the Apple COMMUNICATIONS CARD™.

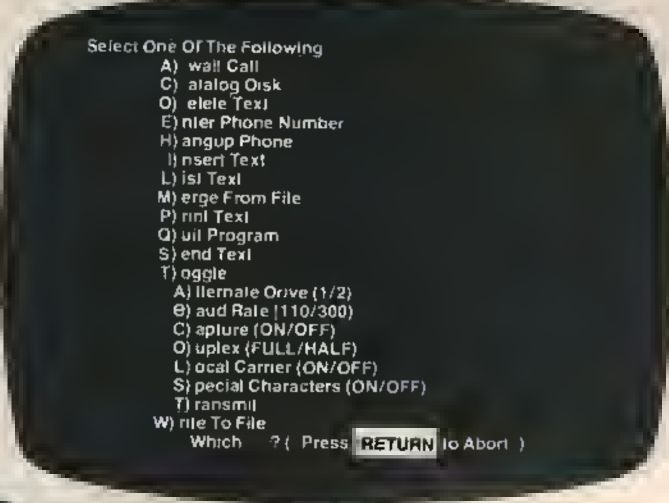
If you use your Apple II™ computer to communicate with other Apples or with timesharing systems, then you need Data Capture 4.0.

If you want to save stock quotes, airline schedules, electronic mail, or other data from a remote computer system for future use, then you need a smart terminal program for your Apple II™. Data Capture 4.0 is the most user friendly and most flexible program of this type. It is specifically designed to take advantage of all the features of the Micromodem II™, but it works equally well with the Apple Communications Card™ and several other popular serial interface cards.

Data Capture 4.0 is the only Apple II™ smart terminal program available which is fully copyable and modifiable. This means that you don't have to worry about backup. Go ahead and make all the backup copies you need.

The unprotected format and helpful documentation make modifications to Data Capture 4.0 very easy to perform. In addition, full technical support is available by phone from Southeastern Software.

Data Capture 4.0 has many other features. Incoming data files are automatically captured regardless of length. Data in the memory buffer can be viewed, edited, printed, saved to or loaded from disk, or transmitted to the remote system at any time. An unattended mode of operation is provided so that you can call your Apple from another location and send data to it or load data from it. Data Capture 4.0 is fully compatible with the Apple III™ in emulation mode. It is also compatible with all popular lower case adapters for the Apple II™, including the widely used shift key modification. An automatic logon utility for use with the SOURCE is provided with Data Capture 4.0. Also included is a HELP text file containing the latest tips and suggestions on using Data Capture 4.0 effectively (many of these are the result of feedback from customers).



See your local Apple dealer today for a demonstration of Data Capture 4.0 or order direct from Southeastern Software at the address below.

(Please include $2.50 for postage and handling.)

If you presently own a previous version of Data Capture you may upgrade to the new version for the difference in price plus $2.50 for postage and handling. MASTER CARD or VISA orders may be placed by phone.

DATA CAPTURE 4.0 .... $65.00
DATA CAPTURE 4.0 / 80* $90.00

*Specify either Videx, Smarterm, Double Vision or Sup 'R' Terminal version

— Dealer Inquiries Invited —

Requires DISK II™, Applesoft II™ and 48K of Memory

DATA CAPTURE 4.0© 1980-Southeastern Software

*Apple™, Apple II Plus™, Disk II™ and APPLESOFT II™ are trademarks of Apple Computer Company.

SOURCE – Source Telecomputing Corporation.

*Micromodem™ is a trademark of D.C. Hayes Associates, Inc.

Southeastern Software

6414 Derbyshire Drive • New Orleans, Louisiana 70126

(504) 246-8438 or (504) 246-7937

master charge THE INTERBANK CARD

BankAmericard VISA

NO. 44 JANUARY 1982

MICRO™

THE 6502/6809 JOURNAL

STAFF

DEPARTMENTS

TUTORIAL / OVERVIEW

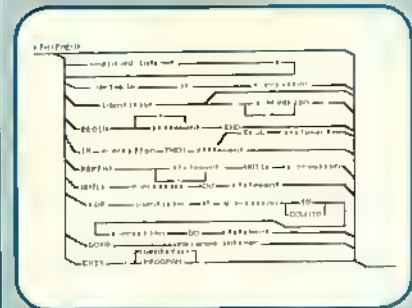
I/O ENHANCEMENTS

PASCAL

SHORT SUBJECTS

COLUMNS

About the Cover



The photo depicted on the cover is of the ancient Roman city of Pompeii. Located at the foot of Mt. Vesuvius, Pompeii was destroyed, yet preserved, by an eruption of that volcano in 79 AD. The Roman architecture in those days depended heavily on the use of blocks, grouped together to form buildings and villas. Similarly, the present-day programming language Pascal takes advantage of block structure to form logical, well-defined programs. Could the Romans have invented Pascal? (For the answer, see PET Vet.)

Cover photo by Ford Cavallari.

Graphic produced with the cooperation of Computer Mart of Nashua, NH.

MICRO is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional entry pending at Framingham, MA 01701
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to

MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

MICRO Editorial

Well before the days of microcomputers, a new programming language entered the growing ranks of software options. This language was not altogether revolutionary; it embodied many concepts already available in other languages. And, it did not pretend to be inexhaustible or all-purpose, as did some of its contemporaries, the monoliths known as FORTRAN V, PL/I, and Algol-68. This new language was designed to be simple, yet complete, and concise rather than verbose. The new entry's name was Pascal.

The Pascal language did not gain immediate popularity. It had no world council or multi-national corporation backing its implementation. However, it did have a beauty, a structured simplicity, which attracted many top educators and analysts in the field of computer science. Thus, Pascal achieved quick acceptance within educational and research communities. California's University of California at San Diego (UCSD) was one of the earliest institutions to latch onto Pascal. Pascal's small size and straightforward implementation ensured its availablity to other communities. But, Pascal's foothold remained in the institutional communities only.

The advent of microprocessor technologies changed the character of computers — from necessarily large to conveniently small. BASIC was picked as the high-level language of choice for the micros, partially because of its small size. Notably absent from the first-generation microcomputers were implementations of the monoliths — they were not feasible. Among the more advanced, structured languages, Pascal alone could be completely and efficiently implemented on a micro. Today, Pascal is available on virtually all 6809-based machines and on most of the 6502-based computers as well.

Pascal boasts many features which set it apart from the primitve BASICs found on most machines. Pascal embodies the four basic control structures which eliminate the need for GOTO statements, and hence line numbers. These structures, known as the *sequence* (a line or sequence of statements), the *if-then-else* conditional, the *do-while* loop, and the *for-next* loop, form the basis for Pascal. See the "Precision Programming" article in MICRO (42:06) for a complete discussion of these constructs. Using these structures, program development time is reduced and code readability is enhanced.

Pascal is also a procedure-oriented language. Analogous to the sections of BASIC code called by the GOSUB statement, procedures are used in Pascal to separate logically distinct functions and to perform repetitive tasks without duplicating code. The advantage over BASIC is that procedures are defined with a list of formal parameters. Thus the procedure is able to modify only those variables which the programmer explicitly instructs it to modify. This feature, almost a necessity for larger program development, is not available in most BASICs.

The final major advantage of Pascal is its strong but flexible typing of variables. Each variable used in a Pascal program must be declared as a type. The standard types are Boolean, integer, character, and real. Further types may be user-defined. All these types may be combined into a structured type; i.e. an array or a record. The "Pascal Tutorial" article in this issue (page 85) discusses variable types, as well as procedures.

The availability and popularity of Pascal is sure to increase as time goes on. Apple Computer has been supporting UCSD Pascal for two years. Commodore has recently announced availability of Pascal on their SuperPET and standard PET computers. The popular FLEX and OS-9 operating systems, built around the 6809, both offer implementations of standard Pascal. Even IBM has opted for Pascal over its own PL/I on their personal computer. The development of ADA, a large Pascal-like language, has also boosted Pascal's attractions. ADA is much larger than Pascal, and will be well-suited to large computer implementation. The more succinct Pascal will most probably remain the microcomputer counterpart of ADA.

MICRO's coverage of languages will continue next month with FORTH. More Pascal material will be appearing in later issues. And, of course, coverage of the microcomputer workhorses, BASIC and machine language, will continue in each issue of MICRO. But for now, turn to the Pascal feature and survey the benefits of structured programming.

Ford Cavallari

MICRO Letterbox

On Games

Dear Editor:

A few notes on your editorial comments about games.

This is a nation of game players, particularly if the definition is broadened to include all forms of escapism. This does not seem to me an unreasonable amplification. Television watching alone must consume enough hours to awe even Carl Sagan. And watching television is an activity usually lacking in any social value. Producers of television fare have learned to produce a mind-deadening, sense-tickling product which does little other than murder time.

Nor is television the only enterprise dedicated to making the days pass. Much of sports, motion pictures and what passes for literature these days is equally dedicated to this end.

Games too, of course, serve to mute the ticking clock. But games at least require some participation. Games are the almost universal introduction to personal computers, and thus have the potential for drawing minds into the realm of computing, where they may be refreshed and expanded by the whole range of the computer's capability. There is, it seems to me, a real place for games.

Whether that place is being realized is, however, an open question. Much of the current crop of games has no more real value than a drugstore's crop of paperbacks: lurid covers and dull interiors. The tack taken to protect this software denies the purchaser it's real value; not the running of the program, but the reading of it. And the understandably sympathetic treatment software developers have received from their special medium denies the customer any voice.

I seem, finally, to have come to the conclusion that computer games represent another failed medium. Again, a medium of great potential power has been perverted to provide cheap thrills and Hi-Res graphics. Something seems to be wrong with our level of maturity. Maybe you're right.

Bob Crafts
Edgartown, MA

Dear Editor:

Your editorial regarding games for computers is straightforward and to the point. Software vendors, especially, often seem to be top-heavy with games.

In my own opinion, though, total disregard for games is a bit heavy handed. Please consider the following:

1. The "tools" for classic games such as Chess and Bridge are simple devices, yet these games have, for centuries, provided challenge and enjoyment. When the tool used in a game is a computer, the possibilities become mind-boggling.

2. Modern quality control statistics is an outgrowth of gaming mathematics (not the other way around). A good game program can likewise teach good programming technique and computer capability, as well as entertain.

3. Recreation, in itself, is socially redeeming. Computers labor hour after hour without complaint; but as a human, I enjoy an hour of Star Trek after work.

Perhaps you could compromise and put in one game per issue. You would still have plenty of room for more serious articles.

Jim Haboustak
South Euclid, OH

What's What in "What's Where"?

Dear Editor:

Last week I received my copy of *What's Where in the Apple*. It's a great book and a valuable source of reference material, but something is missing!

There is no explanation for the codes used in the \Use-Type\ column. What is \HB\? What is \P2\?

Ray Badowski
Stratford, CT

Editor's Note: The following guide will help clarify the Use-Type symbols.

"What's Where" Use-Type Guide

/SE/

1st letter — type information
2nd letter — usage/length information

Type Codes:

S — Subroutine
P — Parameter
H — Hardware
B — Buffer

Usage/Length Codes:

E — Entry
B — Block
n — n-Byte Long
L — Label
F — Flag

Some Common Combinations:

P1:	1-Byte Parameter
Pn:	n-Byte Parameter
PB:	Parameter Block
H1:	Hardware Location
HB:	Hardware Block
FF:	Hardware Flag
SE:	Subroutine Entry Point
SB:	Subroutine Block
SL:	Subroutine Label
BB:	Buffer Block

(Continued on page 47)

Atari 800 Player/Missile Graphics

"Player/Missila Graphics" can be appliad to mora than players and missilas. Tha author explains all and providas utility programs for the Atari 800.

Mike Dougherty
7659 West Fremont Ave.
Littleton, Colorado 80123

The hardware for the early Atari computing systems was designed for the video arcade market. To compete in this market, Atari made use of some very specialized chips. These video chips are able to combine several images on a television raster image simultaneously through direct memory access (DMA) techniques. Fortunately, for Atari computer owners, the Atari 800 hardware is basically the same. Thus, whatever an Atari raster-based arcade game does, the Atari 800 can do. The trick lies in learning bow to program these specialized video chips.

Hardware

A comprehensive discussion of the Atari 800 Personal Computer System may be found in the two manuals available from Atari, *Operating System User's Manual* and *Hardware Manual*. Tbis article will concentrate on the graphics mode called "Player/Missile Graphics."

A player or a missile is a small pattern that can be moved rapidly across the television screen. This terminology was first used when these small objects represented aircrafts, bullets, rockets, etc., in the original Atari video games. The usage of players and missiles, of course, is not limited to games.

Player/Missile Graphics allows the final television image to be constructed from a main background bit map of normal display memory (e.g. GRAPHICS 8) superimposed with up to four players (Player0, Player1, Player2, and Player3) and four missiles (Missile0, Missile1, Missile2, and Missile3). Each player consists of a bit map one byte (8 bits) wide and 128 or 256 bytes high. Each missile consists of a bit map two bits wide and 128 or 256 bytes high. (Different Player/Missile Graphics modes use different amounts of memory, but in either case, the full vertical extent of the television screen may be covered.)

Every player and corresponding missile may be set to a separate Atari color, independent of the normal background color. Thus, by setting a bit to one in a player or missile bit map, that color will be displayed on the television screen. By setting a pattern of bits to one in a player or missile bit map, a Player/Missile Graphics object is created.

Player/Missile Graphics may be constructed on either a "single line playfield" or a "double line playfield." The single line playfield maps each player byte (two bits for a missile) onto one line of the television image, requiring 256 bytes for the complete bit map. The double line playfield maps each player byte (two bits for a missile) onto two consecutive lines of the television image, requiring only 128 bytes for a complete bit map. In addition, both double and single playfields may be one of three overall television screen widths: narrow, regular, or large. Each width affects the amount of memory required by graphics modes.

The width of individual Player/Missile objects may be normal size (GRAPHICS 8 resolution), twice normal size, or four times normal size. Each player may be set to any allowed size, independent of the other player's size. However, all of the missiles must be the same size. If all four players are set to four times normal size and lined up side by side, the total width is that of the narrow playfield. In addition, all four missiles may be treated as a single fifth player — lined up this way they will cover a standard playfield.

Table 1: **Memory Map of Player/Missile Graphics**

Playfield Mode	Offset to Base	# Bytes	Function
Single Line	+0	768	Unused by Player/Missile
	+768	256	All 4 missiles, 2 bits each
	+1024	256	Player0 bit map
	+1280	256	Player1 bit map
	+1536	256	Player2 bit map
	+1792	256	Player3 bit map
Double Line	+0	384	Unused by Player/Missile
	+384	128	All 4 missles, 2 bits each
	+512	128	Player0 bit map
	+640	128	Player1 bit map
	+768	128	Player2 bit map
	+896	128	Player3 bit map

Player/Missile Graphics enforce a programmable "image priority" scheme to handle the case of overlapping player bit patterns. If Player0 has a higher priority than Player1, Player0 will overlap (hide) Player1 when the two bit patterns collide. The display chips also contain specialized collision registers to keep track of which players have overlapped or collided. These specialized hardware functions of the display chips allow the program the luxury of ignoring some very difficult problems. Specifically, as two player bit maps cross paths, no images have to be redrawn after the collision, and the background and players will maintain their programmed shapes.

During the collision, the overlapped image will be automatically generated by the hardware. Further, due to the collision registers, the program does not need to compute if arbitrarily irregular images overlap — simply check the appropriate bit in the appropriate collision register.

The horizontal positions of the players and missiles are determined by a set of horizontal registers. To move a player or missile to a specific horizontal position on the television image, the desired position is POKEd into the appropriate horizontal position register. Vertical movement must be accomplished by shifting the actual bit map in the player's or missile's display memory, up or down the desired number of bytes. Rapid vertical movement may be accomplished through assembly level programming.

To use Player/Missile Graphics, the following steps must be performed:

1. Memory space for the Player/Missile bit maps must be reserved. For single-lined playfields, 2048 bytes are required, beginning on an even 2K address boundary. For double-lined playfields, 1024 bytes are required, beginning on an even 1K address boundary.

2. The page number (high order byte of the address) of that reserved memory must be POKEd into the Player/Missile Page Pointer to define to the hardware where the bit maps are located.

3. Individual player and missile patterns must be POKEd into the appropriate bit maps as detailed in table 1: a bit equal to 1 turns on a pixel, a bit equal to 0 turns off a pixel.

***Table 2:* Register Description of Atari 800**

Address	Player	Function
53248 ($D000)	Player 0	POKE horizontal position of Player/
53249 ($D001)	Player 1	Missile bit map.
53250 ($D002)	Player 2	
53251 ($D003)	Player 3	(POKE will set horizontal register
53252 ($D004)	Missile 0	and PEEK will read collision
53253 ($D005)	Missile 1	registers described next.)
53254 ($D006)	Missile 2	
53255 ($D007)	Missile 3	
53256 ($D008)	Player 0	POKE size of displayed bit map
53257 ($D009)	Player 1	0 - normal size
53258 ($D00A)	Player 2	1 - twice normal size
53259 ($D00B)	Player 3	2 - normal size
53260 ($D00C)	All Missiles	3 - four times normal size
53248 ($D000)	Missile 0	PEEK collision with Playfield
53249 ($D001)	Missile 1	PEEK collision with Playfield
53250 ($D002)	Missile 2	PEEK collision with Playfield
53251 ($D003)	Missile 3	PEEK collision with Playfield
53252 ($D004)	Player 0	PEEK collision with Playfield
53253 ($D005)	Player 1	PEEK collision with Playfield
53254 ($D006)	Player 2	PEEK collision with Playfield
53255 ($D007)	Player 3	PEEK collision with Playfield
53256 ($D008)	Missile 0	PEEK collision with Player
53257 ($D009)	Missile 1	PEEK collision with Player
53258 ($D00A)	Missile 2	PEEK collision with Player
53259 ($D00B)	Missile 3	PEEK collision with Player
53260 ($D00C)	Player 0	PEEK collision with Player
53261 ($D00D)	Player 1	PEEK collision with Player
53262 ($D00E)	Player 2	PEEK collision with Player
53263 ($D00F)	Player 3	PEEK collision with Player
53278 ($D01E)	--- all ---	Reset all collision registers by POKEing 0 into this register
704 ($02C0)	Player/Missile 0	POKE color of 1 bit in displayed bit
705 ($02C1)	Player/Missile 1	map: color = 16*hue +
706 ($02C2)	Player/Missile 2	luminence, defined by Atari
707 ($02C3)	Player/Missile 3	BASIC setcolor command.
708 ($02C4)	Playfield 0	POKE color of 0 bit in bit map
709 ($02C5)	Playfield 1	
710 ($02C6)	Playfield 2	
711 ($02C7)	Playfield 3	
712 ($02C8)	Background	
559 ($022F)	--- all ---	POKE to specify Player/Missile graphics mode:
		61 - narrow playfield
		62 - regular playfield
		63 - large playfield
		45 - double line narrow playfield
		46 - double line regular playfield
		47 - double line large playfield
54279 ($D407)	--- all ---	Page of Player/Missile display memory

Let the base address of the Player/Missile display memory be:
B=PEEK(54729)*256

B+0 ($0000)	--- not used ---	Display memory offset for Player/
B+768 ($0300)	All Missiles	Missile playfields 61, 62, 63
B+1024 ($0400)	Player 0	0 bit - color of background
B+1280 ($0500)	Player 1	1 bit - color of player
B+1536 ($0600)	Player 2	
B+1792 ($0700)	Player 3	

(Continued)

Communications Software from the Utility Specialists

What do you really need in good communications software? We think the most important thing is using the equipment YOU have to transfer data easily and accurately. When Ascii Express made its first appearance in 1979, it was immediately recognized as the finest program of its kind available. Since that time, other programs have appeared with one or two flashy new features, but not a single one of them offers as many choices in terms of hardware compatibility and data transfer modes as the SDS line of communications software.

ASCII EXPRESS II™
If a versatile and accurate data transfer program is what you're after, Ascii Express II is for you. It offers more data transfer modes than any other program of its kind. In fact, we have yet to find a computer system it can't be used with.

In addition, Ascii Express II offers a built-in editor that allows you to produce or modify files before and after transfer, a built-in directory for your most frequently called numbers, and keyboard macros which allow you to send entire sign-on sequences or other host commands with a few simple keystrokes. All these features are designed to help reduce on-line time, and that saves you money.

Z-TERM™
For CP/M users, Z-Term offers all the above, and more . . .

- Support of all 80-column boards and external terminals
- Large copy buffer (41K)
- Unlimited download capacity – no file is too large

For special applications, Z-Term will even allow your Apple to emulate most popular types of terminals for maximum compatibility with your system.

Z-TERM "THE PROFESSIONAL"™
The ultimate communications package. This is the finest program we offer, and it is certain to set the standard for all communications software for years to come.

Z-Term "Pro" provides the solid performance that the truly serious user demands. Many of our users find Z-Term "Pro" to be so valuable that it more than justifies the purchase of the Z-80 Softcard.

Z-Term Pro's special features include:

- Support of the new Novation Apple CAT modem
- Support of specialized transfer protocols, e.g. PAN and Christensen
- Automatic answer and send/receive modes for unattended operation
- Special buffer for parallel printers to ensure no lost characters

We are so convinced that all the SDS programs are the finest available that we've compiled a detailed point-by-point comparison of our programs and those of our leading competitors (for your copy, send a stamped, self-addressed envelope to us at the address below). This comparison of more than 50 items clearly shows what our customers have known all along – that SDS software is the finest. Amen.

CP/M is a trademark of Digital Research.
Z-80 Softcard is a trademark of Microsoft.

SDS
southwestern data systems
P.O. Box 582-M
Santee, CA 92071 Tel. 714-562-3670

TRYING TO MAKE CONTACT?

Table 2 *(Continued)*

B+0	($0000)	--- not used ---	Display memory offset for Player/ Missile playfields 45, 46, 47
B+384	($0180)	All Missiles	0 bit - color of background
B+512	($0200)	Player 0	1 bit - color of player
B+640	($0280)	Player 1	
B+768	($0300)	Player 2	
B+896	($0380)	Player 3	
53277	($D01D)	--- all ---	POKE a 3 to enable Player/Missile DMA

Listing 1: **The BASIC Program Listing**

```
1000 REM ...       D D D G E
1001 REM
1002 REM ... DEMONSTRATION OF VIDEO
1003 REM ... PROCESSDRS IN THE ATARI
1004 REM ... COMPUTER SYSTEM
1005 REM
1006 REM ...    by Mike Dougherty
1007 REM
1008 REM
1009 REM
1100 REM ....................................
1101 REM
1102 REM ... READ AND FOKE THE USR FUNCTIONS
1103 REM ... INTO PAGE 6 MEMORY.
1104 REM ... READ THE OBJECT AS HEX STRINGS
1105 REM ... AND CONVERT TO DECIMAL.
1106 REM
1107 REM
1108 DIM BYTE$(2)
1110 ADDR=256*6:REM FREE BASIC MEMORY
1115 GRAPHICS 0:POKE 752,1:POSITION 10,5:PRINT "FDKEing USR functions."
1120 READ BYTE$
1130 IF BYTE$="**" THEN 2000
1140 GOSUB 1500
1150 POKE ADDR,BYTE
1160 ADDR=ADDR+1
1170 GOTO 1120
1500 REM
1501 REM
1502 REM ... CONVERT BYTE$ TO BYTE
1503 REM
1504 REM
1510 BYTE=0
1520 VALUE=ASC(BYTE$(1)):GOSUB 1600
1530 VALUE=ASC(BYTE$(2)):BDSUB 1600
1540 SDUND 0,0,0,0
1550 RETURN
1600 REM
1601 REM
1602 REM ... CDNVERT ASCII TD HEX
1603 REM ... AND ACCUMULATE IN BYTE
1604 REM
1605 REM
1610 IF VALUE<58 THEN BYTE=BYTE*16+VALUE-48
1620 IF VALUE>57 THEN BYTE=BYTE*16+VALUE-55
1630 SOUND 0,BYTE,10,8
1640 RETURN
1700 REM
1701 REM
1702 REM ... OBJECT DATA FOR USR
1703 REM
1704 REM
1710 DATA 68,68,85,D1,68,85,D0,A0,FF
1720 DATA B1,D0,48,C8,81,D0,AA,68
1730 DATA 91,D0,8A,48,C8,D0,F5,68,60
1750 DATA 68,68,85,D1,68,85,D0,A0,01
1760 DATA B1,D0,48,88,81,D0,AA,68
1770 DATA 91,D0,8A,48,88,D0,F5,68,60
1780 DATA **
2000 REM
2001 REM
2002 REM ... PROGRAM THE VIDEO HARDWARE
2003 REM
2004 REM
2010 GRAPHICS 8:REM SET TD MODE WITH SPARE RAM
2020 POKE 752,1:REM ND CURSOR
2030 SETCOLOR 2,0,0:REM NO BACKGROUND
2110 POKE 559,45:REM DOUBLE LINE NARRDW PLAYFIELD
2120 POKE 704,88:REM INITIAL COLORS
```

4. Player/Missile colors, sizes, image priorities, and horizontal positions must be POKEd into the appropriate video chip registers.

5. The Player/Missile graphics DMA must be enabled by POKEing a three into the Player/Missile Enable register.

These video chip registers are detailed in table 2.

A word of caution: use of Player/ Missile Graphics occasionally interacts with BASIC memory causing unexpected Atari behavior. When this occurs, simply turn the power off and on to reset all of BASIC memory. While developing Player/Missile programs, save new versions prior to execution. Once run, BASIC may not operate correctly![1]

Demonstration

The program DODGE, listing 1, demonstrates the use of Player/Missile Graphics and may serve as a basis for practical applications. This program utilizes the four players to form vertically-moving barriers, along with a single missile, to dodge across the screen under joystick #1 control (STICK(0)). Each time the missile collides with a moving barrier, the round is started over with all new player colors. Horizontal missile movement is controlled through BASIC statements, while the vertical barrier movements must be done *via* USR functions, due to the slow speed of the BASIC interpreter.

A major problem with Player/ Missile Graphics is finding free memory on the correct address boundary to locate the player and missile bit maps. This memory should be protected from modification by the program and BASIC. One such method is to use GRAPHICS 8. This graphics mode occupies the top 8112 bytes of memory, including overhead. Of this 8112 bytes, 256 bytes are reserved for the display list, 6400 bytes are reserved for the GRAPHICS 8 image, 1296 bytes are unused, and 160 bytes are reserved for the text window. The trick is to set

[1] Specifically, I have had trouble in adding new variables to a program after executing a Player/Missile Graphics program. Although the variable was recognized, the value could not be changed from zero. Apparently the variable table pointers were inadvertantly modified by my use of Player/ Missile Graphics.

```
2130 POKE 705,88
2140 POKE 706,88
2150 POKE 707,88
2210 SPACE=PEEK(106)-8:REM PAGE # OF FREE RAM (SORT OF)
2220 POKE 54279,SPACE:REM POINT HARDWARE TO IT
2230 POKE 53277,3:REM ENABLE DMA TRANSFER
2310 POKE 53256,1:REM DOUBLE SIZE FOR PLAYERS
2320 POKE 53257,1
2330 POKE 53258,1
2340 POKE 53259,1
2350 POKE 53260,0:REM NORMAL SIZE FOR MISSILE 0
2410 P0=SPACE*256+1024/2:REM PLAYER BIT MAPS
2420 P1=SPACE*256+1280/2
2430 P2=SPACE*256+1536/2
2440 P3=SPACE*256+1792/2
2450 M0=SPACE*256+768/2:REM MISSILE BIT MAP
2510 POKE 53248,96:REM HORIZONTAL POSITION OF PLAYERS
2520 POKE 53249,168
2530 POKE 53250,132
2540 POKE 53251,60
2600 REM
2601 REM
2602 REM ... SET UP THE PLAYER PATTERNS
2603 REM ... BY SETTING ALL 128 BYTES TO
2604 REM ... THE DESIRED PATTERNS.
2605 REM
2606 REM
2610 BYTE=0
2620 FOR PATTERN=1 TO 8
2630 FOR BAR=1 TO 8
2640 POKE P0+BYTE,255
2650 POKE P1+BYTE,255
2652 POKE P2+BYTE,255
2654 POKE P3+BYTE,255
2682 POKE P0+BYTE+8,0
2684 POKE P1+BYTE+8,0
2686 POKE P2+BYTE+8,0
2688 POKE P3+BYTE+8,0
2690 BYTE=BYTE+1
2692 NEXT BAR
2694 BYTE=BYTE+8
2695 NEXT PATTERN
2900 REM
2901 REM
2902 REM ... SETUP SOUND, MISSILE, AND THE
2903 REM ... PLAYER COLOR -- ENTRY POINT
2904 REM ... FOR EACH NEW ROUND.
2905 REM
2906 REM
2910 SOUND 0,10,100,8
2920 POKE M0+60,3:REM INSERT MISSILE PATTERN
2930 MISPOS=50:POKE 53252,MISPOS:REM MISSILE HORZ POSITION
2940 POKE 707,INT(RND(1)*15)*16+8:REM RESET PLAYER COLORS
2950 POKE 704,INT(RND(1)*15)*16+8
2960 POKE 706,INT(RND(1)*15)*16+8
2970 POKE 705,INT(RND(1)*15)*16+8
3000 REM
3001 REM
3002 REM ... MAIN LOOP:
3003 REM ...  MOVE THE MISSILE ACCORDING TO THE JOYSTICK
3004 REM ...  MOVE EACH PLAYER PAIR UP OR DOWN WITH USR
3005 REM ...  IF THE MISSILE HAS COLLIDED, THEN START OVER
3006 REM ... NEXT LOOP
3007 REM
3008 REM
3010 FOR LOOP=0 TO 1 STEP 0
3020 IF STICK(0)=11 THEN MISPOS=MISPOS-4:POKE 53252,MISPOS
3030 IF STICK(0)=7 THEN MISPOS=MISPOS+4:POKE 53252,MISPOS
3040 X=USR(256*6,P0):X=USR(256*6+26,P2)
3050 IF PEEK(53256)<>0 THEN SOUND 0,100,10,10:POKE 53278,0:GOTO 2900
3060 NEXT LOOP
```

Listing 2: **USR Functions to Move Players**

```
                    *=$0600            FREE MEMORY
             ;
             ;      USR FUNCTIONS USED BY DODGE TO MOVE TWO CONSECUTIVE
             ;      PLAYERS (256 BYTES) DOWN OR UP THE TELEVISION SCREEN.
             ;      NOTE THAT MOVING A PLAYER UP IN MEMORY WILL MOVE THE
             ;      DISPLAY DOWN THE SCREEN.  USE:
             ;
             ;            X=USR(1536,address of players)
             ;            X=USR(1562,address of players)
             ;
00D0                PLAY=$00D0         ZERO PAGE POINTER TO PLAYERS
```

(Continued)

the base and mode of the Player/Missile Graphics in such a way as to force the desired bit maps into the 1296 bytes of unused memory. One solution is to set the Player/Missile base at 2K (eight pages) bytes below the top of memory in the GRAPHICS 8 image and use a single line playfield. This will allow the undisturbed use of all four missiles, Player0, and Player1.

A second solution, the one used in DODGE, is to set the Player/Missile base at 2K (eight pages) bytes below the top of memory in the GRAPHICS 8 image and use a double line *narrow* playfield. This will allow the use of all four missiles and all four players. (Specifying a narrow playfield reduces the normal 6400 bytes of GRAPHICS 8 allowing the missile fields to be used.)

In order to move a DODGE player upwards (downwards), 128 bytes must be rotated. Since BASIC is far too slow for this task, two USR functions were POKEd into the page of free memory at \$0600 - \$06FF. The USR function starting at decimal 1536 (\$0600) rotates 256 bytes (two consecutive players) downward on the television screen (upward in memory). The USR function starting at decimal (\$061A) rotates 256 bytes (two consecutive players) upward on the television secreen (downward in memory). By controlling the horizontal position of the four players, an alternating pattern of vertical movement can be established.

By using simple variations of color, size, position, etc., the reader should be able to use DODGE to master Player/Missile Graphics through simple experimentation.

Mike Dougherty graduated from the University of Tennessee in 1977 with an M.S. degree in Computer Science, and is currently working at Martin Marietta Aerospace in Denver, Colorado. His home-based system presently consists of an Atari 800 with 24K bytes of memory, the Atari 410 recorder and the Atari 850 Interface Module for future communication with single-board computers.

Listing 2 *(Continued)*

```
                ;
0600 68         DOWN    PLA                     NUMBER OF USR ARGUMENTS (1)
0601 68                 PLA                     GET ADDRESS OF PLAYER BIT MAP
0602 85 D1              STA     PLAY+1          HIGH ORDER BYTE
0604 68                 PLA
0605 85 D0              STA     PLAY            LOW ORDER BYTE
0607 A0 FF              LDY     #$FF            POINT TO END OF BIT MAP
0609 B1 D0              LDA     (PLAY),Y        SAVE ONTO STACK
060B 48                 PHA
060C C8                 INY                     POINT TO FIRST OF BIT MAP
060D B1 D0      DOWNLP  LDA     (PLAY),Y        SAVE CURRENT BIT PATTERN
060F AA                 TAX                     TEMPORARILY
0610 68                 PLA                     FETCH PREVIOUS BYTE
0611 91 D0              STA     (PLAY),Y        PUT INTO CURRENT POSITION
0613 8A                 TXA                     RECOVER OLD CURRENT
0614 48                 PHA                     PUSH ONTO STACK
0615 C8                 INY                     NEXT PLAYER BIT PATTERN
0616 D0 F5              BNE     DOWNLP          DO ALL 256 BYTES
0618 68                 PLA                     CLEAN UP STACK
0619 60                 RTS                     RETURN TO BASIC
                ;
061A 68         UP      PLA                     NUMBER OF USR ARGUMENTS (1)
061B 68                 PLA                     GET ADDRESS OF PLAYER BIT MAP
061C 85 D1              STA     PLAY+1          HIGH ORDER BYTE
061E 68                 PLA
061F 85 D0              STA     PLAY            LOW ORDER BYTE
0621 A0 01              LDY     #$01            POINT TO BEFORE FIRST
0623 B1 D0              LDA     (PLAY),Y        SAVE ONTO STACK
0625 48                 PHA
0626 88                 DEY                     POINT TO FIRST OF BIT MAP
0627 B1 D0      UPLP    LDA     (PLAY),Y        SAVE CURRENT BIT PATTERN
0629 AA                 TAX                     TEMPORARILY
062A 68                 PLA                     FETCH PREVIOUS BYTE
062B 91 D0              STA     (PLAY),Y        PUT INTO CURRENT POSITION
062D 8A                 TXA                     RECOVER OLD CURRENT PATTERN
062E 48                 PHA                     PUSH ONTO STACK
062F 88                 DEY                     NEXT PLAYER BIT PATTERN
0630 D0 F5              BNE     UPLP            DO ALL 256 BYTES
0632 68                 PLA                     CLEAN UP STACK
0633 60                 RTS                     RETURN TO BASIC
                        .END
```

MICRO

NEW ROM BOARD FOR THE APPLE II*
$125.00 WITH UTILITY ROM.

With Andromeda's new ROM Board, you can plug many useful utility programs into your Apple II. Because ROM memory never forgets, you can access these utilities instantly without having to load them from disk.

The ROM Board comes with the utility ROM, which gives you five powerful options to apply to your Applesoft* programs. With the Utility ROM, you can do automatic line numbering, control a program list with a page mode, restore a crashed Applesoft* program in memory, alphabetize a disk catalogue and create a disk without DOS, giving you an extra 8K on your disk. Any of Soft Control Systems' other ROMS can be used, such as the Dual DOS in ROM, and Your'ple ROM.

You can install 2K PROMS, 4K PROMS, or even 2K RAM chips in each of the two memory sockets. So you can even have the Read - Write capability of RAM to develop PROM Programs yourself, or just have an extra 2K RAM for your machine - Language programs. Two 2732 PROMS allow a total of 8K of memory on the Board.

Now with One Year Warranty.

ANDROMEDA
INCORPORATED
Greensboro, NC. 27410
P.O. Box 19144
919 852-1482

VISA

master charge

Price for Andromeda 16K RAM expansion board now only $120.00. Please add $5 for shipping and handling. North Carolina residents add 4% sales tax.

*DEALER INQUIRIES WELCOME.

Experimenters and the Color Computer

A brief summary of tha normal capablities ot the TRS-80 Color Computar, and an examination of tha unit's I/O capability. Detailed instructions are given for home-brew software expansion *via* the Program Pack port, and a list of vendors of softwere and hardware axpansion provided.

Ralph Tenny
P.O. Box 545
Richardson, Texas 75080

An Overview

Radio Shack's TRS-80C Color Computer is two machines in one: it is a really fine games machine with good color graphics capability, and it is the start of a very powerful, low-cost experimenter's computer. Unless you are really into color graphics, however, the graphics capability could get in your way.

That last statement deserves a bit of explanation. When you first get your Color Computer, it is a real shake-up to enter ?MEM and get the answer "8487". 8K bytes out of 16K? What gives? The answer is that the machine automatically reserves four screens (1.5K per screen) of space for color graphics, whether or not you plan to use *any* graphics! Most of this memory can be recovered using the command "PCLEAR 1". This forces the computer to reserve only one screen for graphics. A second procedure gets it all back:

POKE 25,6;ENTER;NEW;ENTER

As you look at the machine, you will notice that the keyboard isn't a full, professional keyboard. What is there is actually quite good for the overall cost/features trade-off. The back panel of the machine has a number of openings for printer cable, joysticks, RF outlet to the color TV set, and cassette cable. On the right side you'll find a slot to accept the Program Packs, which are Radio Shack's major software distribution media. This same slot is also the computer's expansion port, which is already being taken advantage of by some manufacturers selling expansion hardware packages. Some of these manufacturers and their products are listed later in this article, but the field is expanding so rapidly that publication deadlines prevent inclusion of the newest ones.

One of the major features of the machine is the internal architecture, which is superbly laid out with very cost-efficient design. The details of this architecture have been thoroughly described[1], and most will not he repeated here. The major advantage of the Color Computer may well be the fact that the memory map is software selectable to a great degree. Also, the expansion port has a decode defeat line which allows an external peripheral to re-assign the entire memory map except for the display and I/O hardware which are located above $FF00.

Much of the joy in working with the Color Computer is finding all the built-in "hooks" which were left for future expansion, but I have also greatly enjoyed working with the 6809 processor. As MICRO editor Robert Tripp has already pointed out, the 6809 is really what all us 6502 buffs have wished for but couldn't have until now!

Modification Ideas

Two items are an absolute "must" for anyone contemplating modification of the Color Computer: a copy of Reference 1, and the Service Manual for the Color Computer. The stock number for the Manual is 26-3001/3002, and it most likely will have to be ordered by your local Radio Shack store. The manual gives schematics, service information, and a description of operation for most of the circuits. Even with these resources, I recommend that you not attempt hardware modifications unless you have considerable experience with computers and digital hardware.

One of the most obvious areas for changing the Color Computer is *via* the Program Pack port. New software can be installed in 2716 or 2732 EPROMs, by modifying one of the cheapest Program Packs. It is possible to (carefully!) unsolder the masked ROM present in these packs and replace it with a socket. If this is done, the original ROM should work in the socket, in case you want to restore the original function.

If you compare the EPROM pinouts vs. the socket connections in the Program Pack, you will see that the Intel format 2716's and the 2516 and 2532 EPROMs from Texas Instruments will (almost) drop into the sockets. A few minor etch cuts (summarized in figures 1 and 2) will allow proper operation of these EPROMs in place of the original masked ROM. The etch cuts required for the 2716/2516 (essentially identical parts) are shown in photos 1 and 2. However, if you want to run two EPROMs, some additional address decoding will be required so that the two ROMs do not interfere with each other. This decoding can be done by using a single quad NAND gate as shown in the circuit in figure 3, and further detailed in photos 1 and 2. A few more changes would be required to adapt the Intel 2732 to the address connections used in the Program Pack, and these are detailed in figure 4. Photos 3 and 4 show the modified PC card installed in the Pack, and the Pack installed in the computer.

You need to consider one additional modification to the Program Pack before you can use it without interference from the BASIC ROMs installed in the computer. The regular BASIC (not Extended BASIC) checks for the presence of a Program Pack and vectors into the Program pack if it is installed.

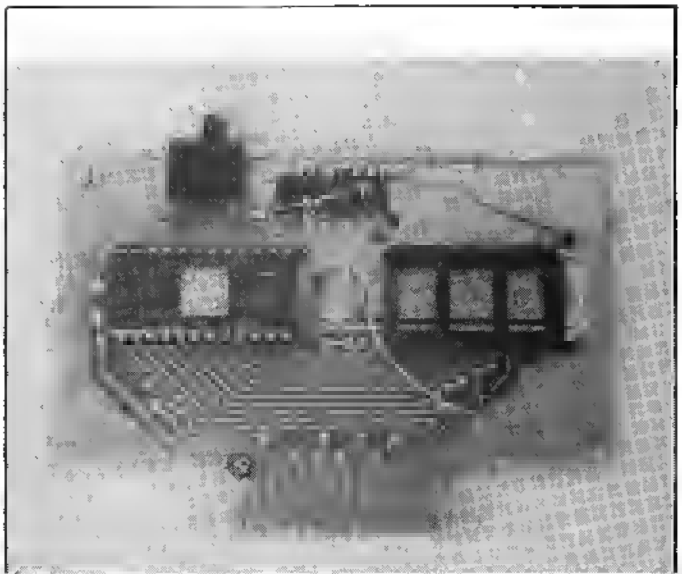


Photo 1: Component side view of Program Pack card, with two-ROM modification, defeat switch and decoding installed. Black circles mark etch cuts; the left one inhibits pack autostart. The other etch cut isolates A12 which, in turn, is used as Block Select by the decoder.

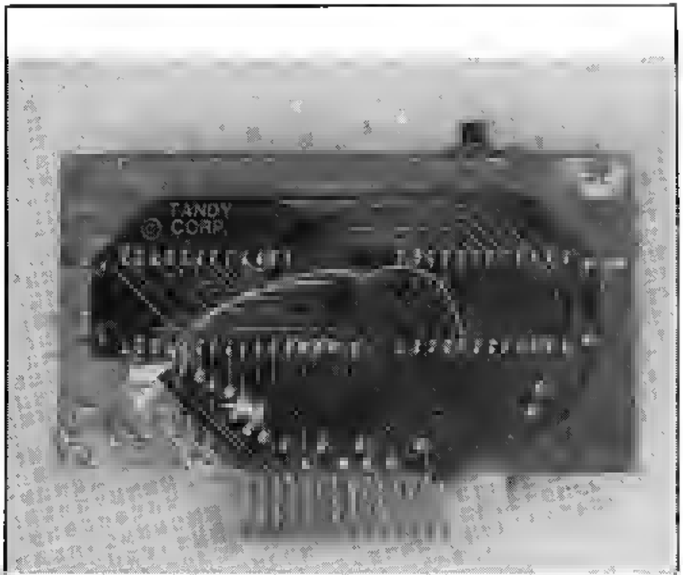


Photo 2: Back side of Program Pack card. White arrows show etch cuts; bottom one isolates all; adjacent jumper grounds pin 18 on ROMs for 2716/2516 use. Restore this connection for 2532 EPROMs. The other two etch cuts isolate incoming decode signal lines to allow separate decoding.

Photo 3: **Modified Program Pack card installed in carrier. Center post in carrier holds screw to secure lid; case also snaps together along edges. Lower half of case must be notched for switch.**

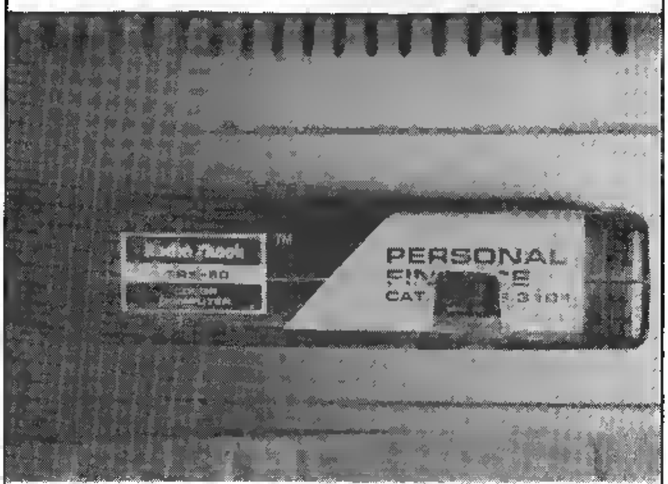


Photo 4: **Program Pack installed; note case cutout and protruding defeat switch handle.**

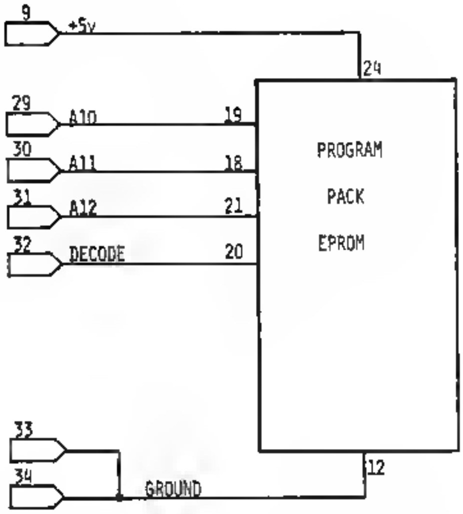


Figure 1: **High order address lines and chip select connections on standard game pack cartridge.**

This is accomplished by interconnecting two pins on the PC board in the Pack. Pin 7 of the port is connected to pin 6 of the port only when the pack is in place. This path must be interrupted if you do not want the Pack to take over automatically. In photo 1, the left-hand card edge pin is circled, indicating the required etch cut on top the board.

The switch shown in this series of photos is shown also in the circuit of figure 3, along with a necessary pull-up resistor. Normally, the Pack can be accessed from BASIC using "EXEC 49152". If this switch is open, entry into the Program Pack is blocked.

External I/O

The Color Computer does not have any kind of external I/O capability except for the ports intended for use with the regular peripherals — printer, cassette recorder and game paddles. If an applications program is placed in the Program Pack, some of these ports can be made available for external I/O of a limited form. Let's examine the I/O capability of these various ports:

Game Paddle Ports — One discrete switch closure to ground and two voltage inputs at each port. Although these voltages must be strictly limited to five volts maximum, the computer will digitize the input with a resolution of six binary bits or 78 millivolts, which is adequate for many applications. Another use for these voltage ports could be to sense mechanical positions, if suitable potentiometers are attached to the moving machinery and powered from pins 5 and 3 of the port.

Printer Port — One RS-232 output line, two RS-232 input lines and ground. The input lines will "read" digital signals, but the output line will swing nearly 24 volts peak-to-peak, and would require an external current-limiting resistor and diode clamping if it were used to drive something such as CMOS logic. This last experiment should not be undertaken without a thorough understanding of circuit design techniques and the limitations of the IC's used, both inside the Color Computer and in the external circuitry.

Cassette Recorder Port — In general, only low-level audio signals can be handled at this port, but you may be able to think of some way to apply such signals. In addition, an isolated relay closure is available for external on-off functions. For the safety of the computer, it would be best to limit this application to circuitry requiring not more than 100 mA of DC current.

(Continued)

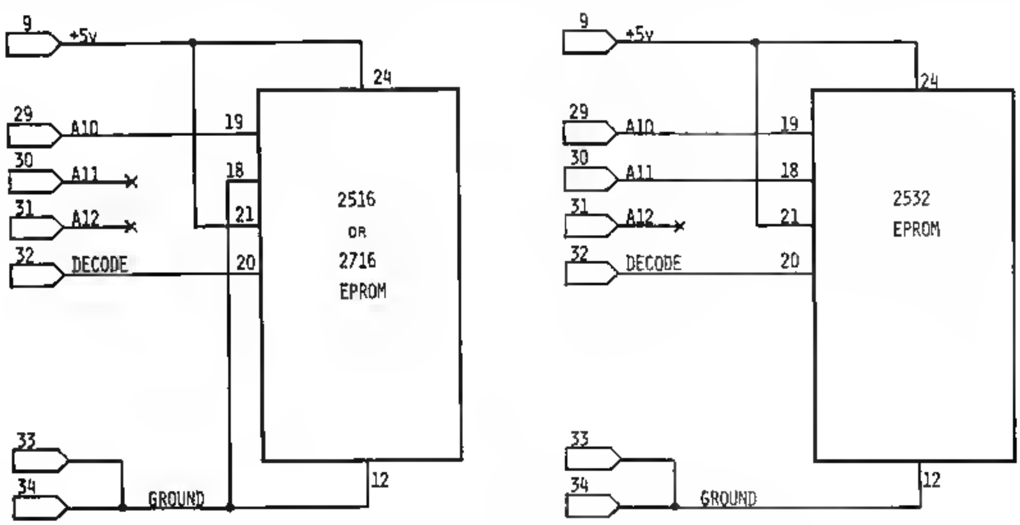
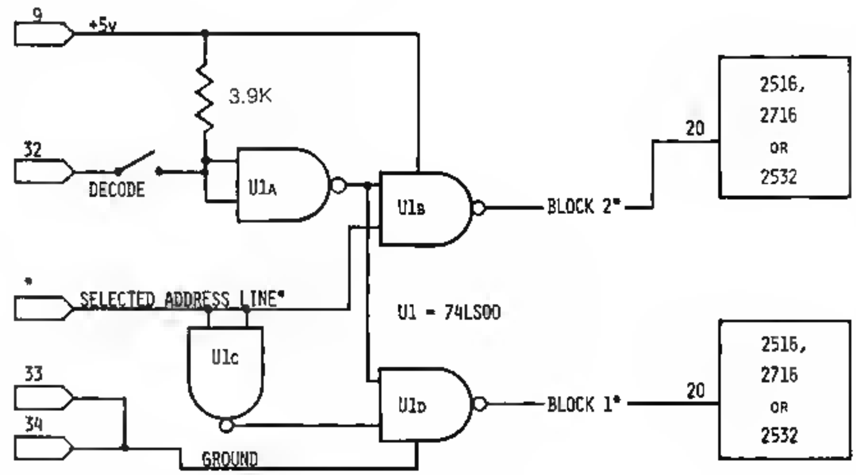


Figure 2: **Connections for single EPROM in Program Pack.**



*If A11 (pin 30) is used, Block 1 = $C000-$C7FF and Block 2 = $C800-$CFFF. If A12 (pin 31) is used, Block 1 = $C000-$CFFF and Block 2 = $D000-$DFFF.

Figure 3: **Schematic of address decoding required to support two EPROMs in one Program Pack.**

Commercial Expansion Hardware and Software

So far, only a limited amount of expansion hardware has been announced, with much of it being related to floppy disk and memory expansion. It is possible to expand the Color Computer to 32K internally, using suggestions from reference 1. However, this will invalidate your warranty, so plan accordingly. The following items are currently advertised in various home computer journals, and a list of addresses for the manufacturers is provided at the end of this article.

EXATRON: Plug-in cartridge containing 32K of dynamic read/write memory and provision to accept a plug-in disk controller. ROM-based software furnished to drive either configuration.

The MICRO WORKS: Memory expansion to both 16K and 32K internal and a serial-to-parallel adapter to drive serial printers. Assembly-language monitor, Disassembler, and software described as a software development system — assembler, editor and monitor in one package.

TALLGRASS: Double-density floppy disk controller, 64K internal memory expansion, ROM-based disk driver + monitor.

SPECTRAL ASSOCIATES: EPROM Programmer, Editor/Assembler, Monitor, Disassembler, games.

NELSON SOFTWARE: Memory conversion kits, Word processor software, games.

SSM, Inc,: Disassembler, Games, software keyboard expansion.

DATASOFT, Inc.: Monitor, Assembler, Disassembler, Debugger, Editor.

Vendor Addresses

Datasoft, Inc.
16606 Schoenborn St.
Sepulveda, CA 92343

Exatron
181 Commercial St.
Sunnyvale, CA 94086

The MICRO WORKS
P.O. Box 1110
Del Mar, CA 92014

Nelson Software Systems, Inc.
P.O. Box 19096
Minneapolis, MN 55409

Spectral Associates
141 Harvard Ave.
Tacoma, WA 98466

SSM Soft Sector Marketing, Inc.
6250 Middlehelt
Garden City, MI 48135

Tallgrass Technologies Corp.
9009 W. 95th St.
Overland Park, KS 66212

Reference

1. "What's Inside Radio Shack's Color Computer?," *BYTE*, March 1981, p. 90.

MICRO

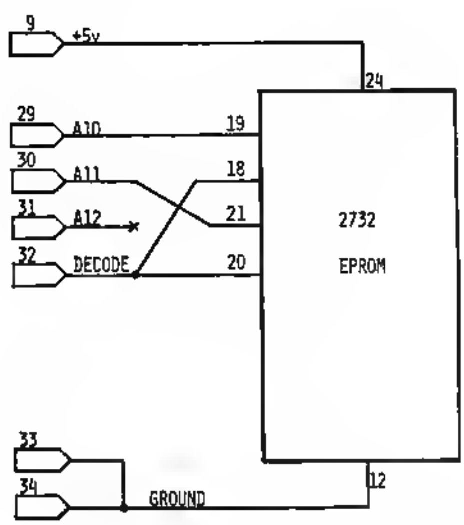


Figure 4: **Address and enable connections required for a single 2732 in a Program Pack.**

OSI TRS-80 COLOR-80 OSI

GALAXIAN · 4K · One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system — A bargain at $9.95 OSI

LABYRINTH · 8K · This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you treck down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! — $13.95. OSI

THE AARDVARK JOURNAL

FOR OSI USERS — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.
2) Complete listings of two word processors for BASIC IN ROM machines.
3) Moving the Directory off track 12.
4) Listings for 20 game programs for the OSI.
5) How to write high speed BASIC — and lots more —

Vol. 1 (1980) 6 back issues · $9.00

Vol. 2 (1981) 4 back issues and subscription for 2 additional issues · $9.00.

ADVENTURES!!!

For OSI, TRS-80, and COLOR-80. These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch".)

Adventures require 8K on an OSI and 16K on COLOR-80 and TRS-80. They sell for $14.95 each.

ESCAPE FROM MARS (by Rodger Olsen)

This ADVENTURE takes place on the RED PLANET. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)

This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

TREK ADVENTURE (by Bob Retelle)

This one takes place aboard a familiar starship. The crew has left for good reasons · but they forgot to take you, and now you are in deep trouble.

DEATH SHIP (by Rodger Olsen)

Our first and original ADVENTURE, this one takes place aboard a cruise ship · but it ain't the Love Boat.

VAMPIRE CASTLE (by Mike Bassman)

This is a contest between you and old Drac · and it's getting a little dark outside. $14.95 each.

OSI NEW-NEW-NEW OSI
TINY COMPILER

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry — 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic — about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -, =, *, /, , <, > , Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.

TINY COMPILER — $19.95 on tape or disk OSI

SUPERDISK II

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing for —

SUPERDISK II $29.95 (5¼") OSI

BARE BOARDS FOR OSI C1P

MEMORY BOARDS!!! — for the C1P — and they contain parallel ports!

Aardvarks new memory board supports 8K of 2114's and has provision for a PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expension connector on the 600 board. Available now!

PROM BURNER FOR THE C1P — Burns single supply 2716's. Bare board — $24.95.

MOTHER BOARD — Expand your expansion connector from one to five connectors or use it to adapt our C1P boards to your C4/8P. · $14.95.

16K RAM BOARD FOR C1P — This one does not have a parallel port, but it does support 16K of 2114's. Bare Board $39.95.

WORD PROCESSING THE EASY WAY— WITH MAXI-PROS

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (0S65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.

MAXI-PROS — $39.95. Specify 5¼ or 8" disk.

SUPPORT ROMS FOR BASIC IN ROM MACHINES — C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. — $39.95

C1E/C2E similar to above but with extended machine code monitor. — $59.95 OSI

ARCADE GAMES FOR OSI, COLOR-80 AND TRS-80 (8K OSI, 16K TRS-80 AND COLOR-80)

TIMETREK · A REAL TIME, REAL GRAPHICS STARTRECK. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! $14.95.

STARFIGHTER · This one man space war game pits you against spacecruisers, battlewagons, and one man fighters, you have the view from your cockpit window, a real time working instrument panel, and your wits. Another real time goody. $9.95

BATTLEFLEET · This grown up version of Battleship is the toughest thinking game available on OSI or 80 computers. There is no luck involved as you seek out the computers hidden fleet. A topographical toughie. $9.95

QUEST · A NEW IDEA IN ADVENTURE GAMES! Different from all the others, Quest is played on a computer generated mape of Alesia. Your job is to gather men and supplies by combat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time.

16K COLOR-80 OR TRS-80 ONLY. $14.95

Please specify system on all orders

This is only a partial listing of what we have to offer. We offer over 120 games, ROMS, and data sheets for OSI systems and many games and utilities for COLOR-80 and TRS-80. Send $1.00 for our catalog.

AARDVARK TECHNICAL SERVICES, LTD.
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110

OSI COLOR-80

Sweet-16 Revisited

The Apple II's Integer BASIC ROM supports a powerful and seldom used pseudo machine known as Sweet-16. In this article, the Sweet-16 instruction set is described and programming hints, using a macro-assembler, are presented. Ways to use Sweet-16 on an Apple II Plus are also discussed.

Charles F. Taylor, Jr.
587F Sampson Lane
Monterey, California 93940

Steve Wozniak had a great idea in Sweet-16. Manipulating 16-bit quantities on an 8-bit 6502 in assembly language is inherently tedious. The Sweet-16 interpreter is a solution to that problem. It presents, to the programmer, a virtual 16-bit machine with a convenient set of instructions and sixteen 16-bit registers. The programmer using Sweet-16 can thus pretend that he or she is programming a 16-bit computer! The interpreter performs the necessary translations.

It appears that Sweet-16 has never really caught on. I base this statement on the general absence I've noticed, of non-trivial Sweet-16 programs in the popular microcomputer magazines. Perhaps the problem is that Sweet-16 code is relatively slow (up to ten times slower than the equivalent 6502 code, according to Wozniak). For many applications, the loss of speed would be unimportant, particularly if the programmer jumps in and out of Sweet-16, using Sweet-16 code only for 16-bit operations.

I suggest that there are several other, more significant reasons why Sweet-16 hasn't caught on.

1. The lack of readily available documentation;
2. The inconvenience of hand-assembling Sweet-16 code; and
3. The fact that the Sweet-16 interpreter is not supplied with the Apple II Plus (as it resided, along with the miniassembler, in the Integer BASIC ROM of the Apple II).

Richard C. Vile, Jr., in his article "Sweet-16 Programming Using Macros" (MICRO 20:25), made a significant contribution by showing how the use of macros (with an appropriate macroassembler) can totally eliminate the inconvenience of hand assembly when using Sweet-16. For those readers unfamiliar with the subject, macros in assembly languages are a bit like abbreviations in ordinary language. The meaning of the macro (a series of instructions or an arbitrary sequence of bytes) is defined at the beginning of a program and associated with an identifier. Then, during assembly, the macroassembler substitutes the expanded meaning of the macro wherever the associated identifier occurs in the source code. Vile used this procedure to permit Sweet-16 programming using a set of mnemonics rather than raw op-codes.

Scope of this Article

In this article I propose to do the following:

1. Add to the store of Sweet-16 documentation, correcting some earlier errors in the process;
2. Provide a corrected set of macros;
3. Illustrate the use of Sweet-16 with a non-trivial programming example ("Quicksort");
4. Show how Apple II Plus owners can use Sweet-16;
5. Show how to use the power of a macroassembler more fully to decode the Sweet-16 mnemonics, thus producing a program which executes considerably faster.

Sweet-16 Described

The Sweet-16 virtual machine consists of sixteen 16-bit registers, R0-R15. These are implemented in the first 32 bytes of page zero (locations $00-$1F). Page zero was chosen to take advantage of the 6502's zero page addressing modes, and the greater speed of zero page operations. R0 is at locations $00-$01, R2 at $02-$03, and so on. The virtual machine architecture is illustrated in figure 1. All data in the registers is stored low byte first; i.e., R0L (the low-order byte of R0) is at location $00, and R0H (the high-order byte) is at location $01. This permits the contents of any register to be treated as an address, consistent with the usual 6502 storage convention. All arithmetic operations are implemented with this in mind.

Several registers have special functions. R0 serves as a 16-bit accumulator. R15 serves as the program counter for the Sweet-16 interpreter. R12 serves as a stack pointer for Sweet-16 subroutine calls. R13 serves as a result register for the CPR (compare) instruction. The high-order byte of R14 (R14H) is used as a status register to point to the last register affected, and to store the carry bit for use by conditional branching instructions. R1 through R11 are general-

Figure 1: **Sweet-16 Architecture**

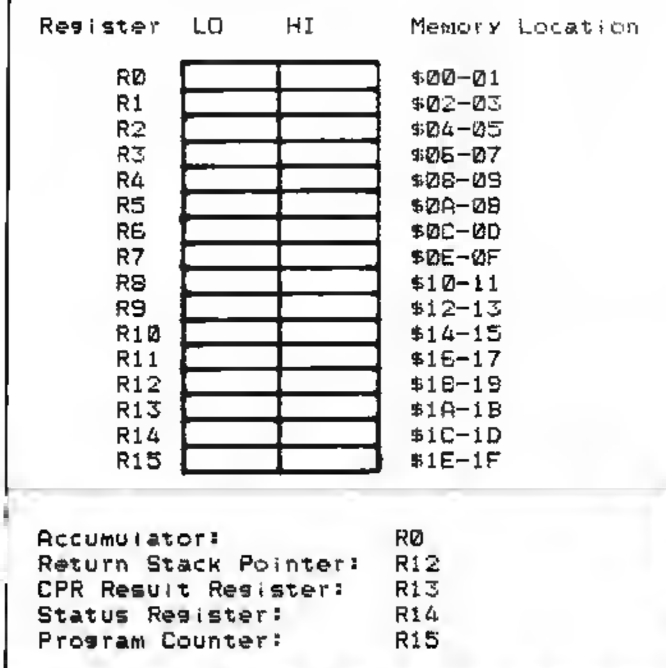
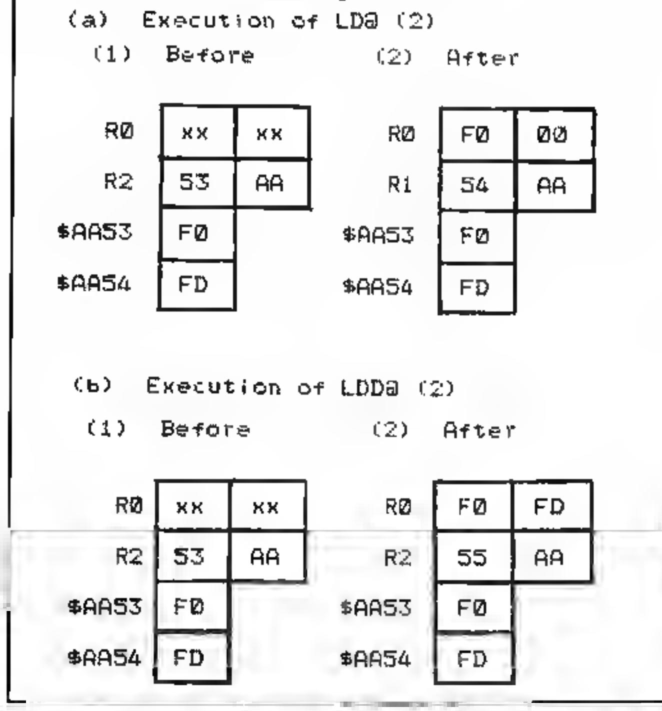


Figure 2: **Auto-Incrementing**



purpose registers and may be used for holding data, addresses, or user-defined stack pointers.

Sweet-16 has three basic addressing modes: immediate, register direct, and register indirect. There is only one instruction which uses the immediate mode: the SETR instruction. SETR (2 $1234), for example, stores the quantity $1234 in R2. The register direct instructions each specify a register as the operand, and act upon that register and possibly also upon the accumulator (R0). Examples are LD (3), which takes the contents of R3 and loads them into the accumulator, and ST (3), which does precisely the opposite. The arithmetic instructions are also register direct. ADD (4), for example, adds the contents of R4 to the contents of R0 and leaves the result in R0. INCR (5) increments the contents of R5 by one.

In register indirect addressing, the operand is not a register, but an address pointed to by a register. Before a memory location can be accessed, its address must be loaded into a register. For example, the sequence of instructions

```
SETR   (2 $AA53)
LD@    (2)
```

would load the contents of memory location $AA53 into R0L (and would set R0H to 0). This is an example of an 8-bit operation. The instructions

```
SETR   (2 $AA53)
LDD@ (2)
```

would load the contents of memory locations $AA53 and $AA54 into R0. This is, of course, a 16-bit operation.

A distinctive characteristic of these register indirect instructions is that they provide automatic incrementing of the register containing the address. After the LD@ (2) instruction, for example, the contents of R2 will be $AA54, the address of the next 8-bit quantity. After the LDD@ (2) instruction, the contents of R2 will be $AA55, the address of the next 16-bit quantity. This auto-increment feature is usually convenient, especially since many operations involve sequential memory accesses. Figure 2 illustrates the effects of these instructions.

The Sweet-16 branching instructions are conventional except that all branches, including the unconditional branch and the subroutine call, are relative. Although this restricts the range of branches to between −128 and +127 bytes, it is not a serious restriction because Sweet-16 code is so dense. In any case, an absolute jump can be simulated by storing the destination address (less one) in the program counter (R15). (It is necessary to subtract one from the destination address because Sweet-16, like the 6502, increments the program counter before fetching the next op-code.)

The Sweet-16 instruction set makes it easy to implement stacks. In fact, three different kinds of stacks are directly supported, and any register can be used as a stack pointer (although I wouldn't try R14 or R15!). The three types of stacks are as follows:

1. An 8-bit stack which grows upward in memory, which I will call "type 1." The sequence

   ```
   ST@ (Rn)
   POP@ (Rn)
   ```

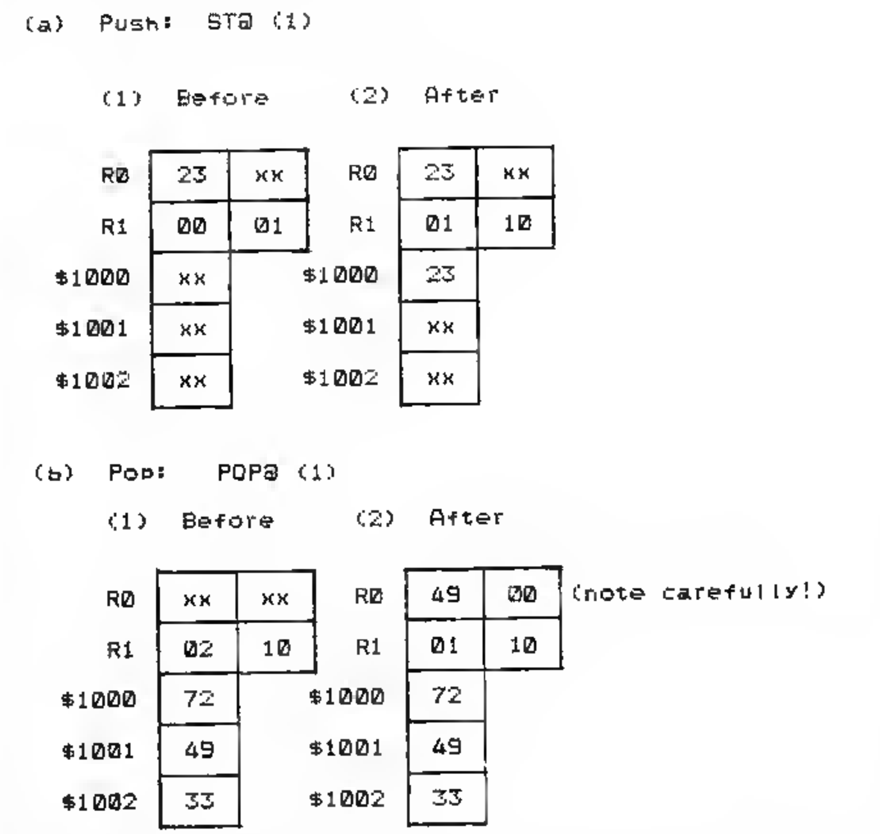
   alternately pushes the contents of R0L onto the stack pointed to by Rn and pops the value from the top of the stack to R0L (setting R0H to 0). The stack pointer is always left pointing to the next available (vacant) stack location. Figure 3 illustrates these actions.

2. A 16-bit stack which grows upward, which I will call type 2. Its operation is illustrated by example in figure 4. Note that its pointer, like that of the type 1 stack, is always left pointing to the next available stack location.

3. An 8-bit stack which grows downward in memory, which I will call type 3. Its operation is illustrated by example in figure 5. Note that, in contrast to the other two types, the type 3 stack pointer is always left pointing to the "top" element of the stack, not the next available location.

The Sweet-16 instruction set is summarized in table 1. Parentheses are used around each operand because they

Figure 3: Stack Operations (Type 1)



are required by the syntax of the macroassembler. Parentheses in the "effects" column are used to represent indirect addressing. The symbol ": =" is used to mean "is replaced by." The symbol "Rn" is used to represent any register, R0-R15. The expression "R0L : = (Rn)" thus means that the contents of the memory location pointed to by Rn is loaded into R0L, the low-order byte of the accumulator. Each entry in the table has been verified by stepping through the code.

Sweet-16 Macros

Suppose that we want to, as discussed before, load the contents of locations $AA53 and $AA54 into the Sweet-16 accumulator. By consulting table 1, we see that the proper sequence of bytes to accomplish this is

```
12 53 AA 62.
```

The "12" is the op-code for SETR when the affected register is R2, the "53 AA" is the address in low-high format, and the "62" is the op-code for the LDD@ instruction when the register affected is R2. The Sweet-16 macros allow us to write instead

```
SETR   (2 $AA53)
LDD@  (2).
```

The macroassembler then takes care of substituting the appropriate Sweet-16 machine code for the mnemonics. This certainly makes life simpler! The savings are even greater for the relative branch instructions, because the macroassembler can take care of computing the relative displacement from the addresses.

(Because I use the ASSM/TED Macroassembler from Eastern House Software, I will use its mnemonics in the following discussion.)

There was, unfortunately, one serious error in the previously published set of Sweet-16 macros. The RELBR (relative branch) defined there does not compute relative displacements correctly for forward branches. Conditional assembly was used to compute displacements separately for reverse and forward branches. The displacement for reverse branches was given as "LOC – = – 1", which is correct. "LOC" is a parameter of the macro and is the destination address. The "=" stands for the current location. The displacement given for forward branches was "= – LOC + 1", which cannot be correct because it is a negative quantity. The correct forward displacement is "LOC – = – 1", which is the same expression as that for reverse branches. Conditional assembly is, therefore, not needed. The correct version of the macro is as follows:

```
!!!RELBR  .MD (LOC)
          .BY LOC – = – 1
          .ME
```

That this is the correct computation for both forward and reverse branches can be verified by stepping through the code. You should note that, at the time of the branch, the program counter is pointing at the displacement byte. Following the branch, the program counter should point to one byte before the destination address.

One other macro in the set needs to be modified slightly. The "BRK" macro is assembled to op-code "00" (the 6502 BRK op-code, but the Sweet-16 RTN op-code). This is easily fixed by changing the mnemonic to "BK" to avoid confusion with the 6502 mnemonic.

The corrected set of Sweet-16 macros is shown in listing 1, at the beginning of the demonstration program, which will be discussed soon. The reason that the "@SW16" macro shows address "$7689" rather than "$F689" will be explained later in the article.

Sweet-16 Demonstration Program

In selecting a demonstration program, I wanted to choose a program that demonstrates some of the characteristic features of Sweet-16, and at the same time, is perhaps useful. The Quicksort algorithm, invented by the eminent British computer scientist C.A.R. Hoare, seemed an ideal choice.

Quicksort is usually implemented in a language such as Pascal, using recursion. Unfortunately, recursion is not available to us in low-level languages such as 6502 assembly language or Sweet-16, so we must simulate it. The simulation of recursion in this case requires a 16-bit stack. The algorithm also requires numerous 16-bit comparisons and data manipulations. As Sweet-16 makes the implementation of all of these operations easy, it is a good choice for this application.

The specific problem to be solved is to sort a list of 16-bit addresses into ascending order. Briefly, here is how the program works: suppose the first element in the list is located at address L and the last element at address R. We choose an element from the list, say the element located at address (L + R)/2, and call its value X. Starting from

Figure 4: Stack Operations (Type 2)

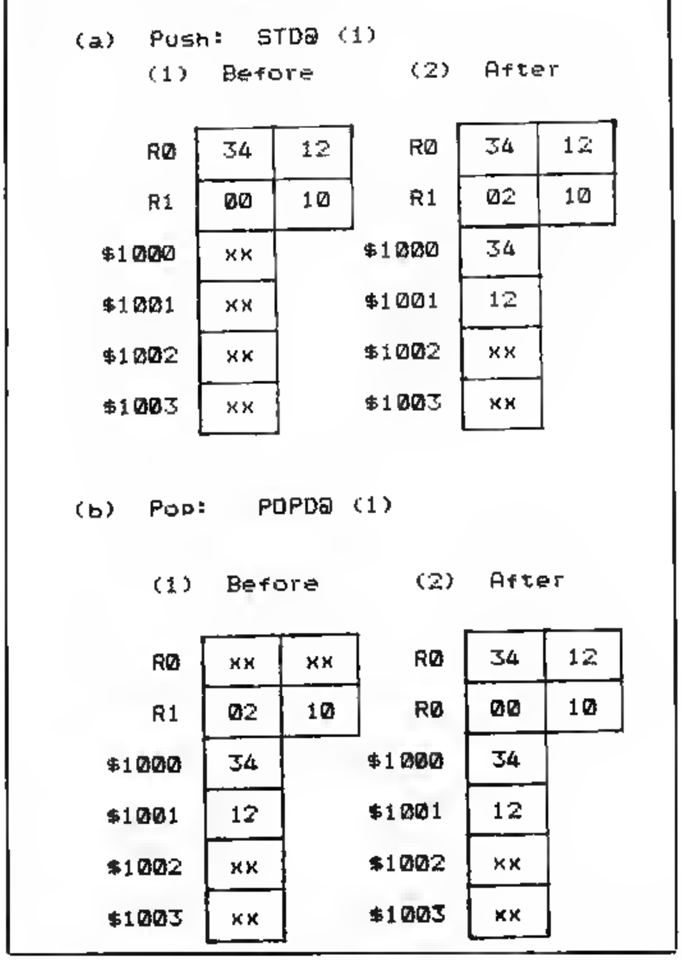
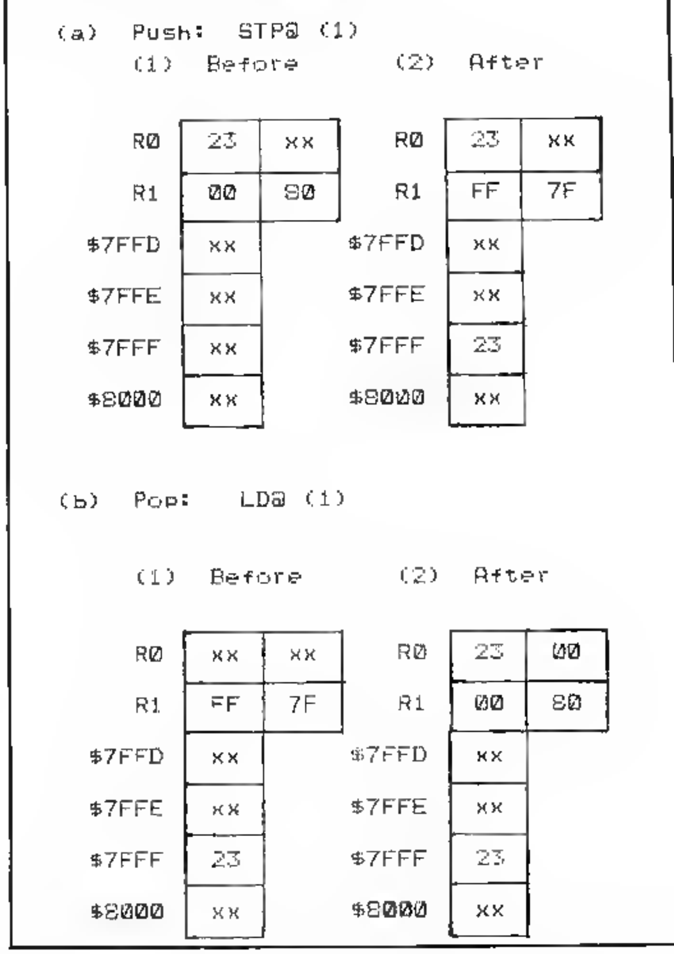


Figure 5: Stack Operations (Type 3)



L and moving toward the center of the list, we look for an element which has value greater than or equal to X. Then we start from R and move toward the center of the list until we find an element with value less than or equal to X. These two elements are then exchanged. We then continue moving toward the center from both sides, exchanging elements to the left of X that are greater than or equal to X, with elements to the right of X that are less than or equal to X.

When our paths cross, the list is said to have been partitioned about X. Note that X itself can be moved. When the partition is complete, all values to the left of X will be less than or equal to X, and all values to the right of X will be greater than or equal to X. In addition, X will have migrated to its proper position in the list and need not be considered further. We thus have two sublists to be sorted.

As we can consider only one list at a time, we put the right-hand sublist aside for later consideration. We do this by pushing its boundaries onto the stack. The left-hand sublist is then partitioned in the same manner as the original list. Eventually the sublist being partitioned will be so short that it is trivially sorted. Then we pull the addresses of another sublist from the stack and continue to partition it until it too is sorted. If we repeat this process until the stack is empty, the result is a sorted list.

Quicksort is, on the average, one of the fastest sorting algorithms known. The number of operations (comparisons and exchanges) required to sort a list of N elements is of the order of N times the base 2 logarithm of N. Compare this to the ubiquitous Bubblesort, which requires an average of N-squared operations. The advantage of Quicksort thus increases rapidly with the size of the list to be sorted.

The program is shown in listing 1. It may appear to be quite long, but remember that the macros only have to be entered into your system once; you may use them for as many programs as you like without reentering them. Excluding the macros, comments, and assembler directive, the program is implemented in 74 lines of code. An equivalent, non-recursive, implementation in Pascal took 38 lines. I leave to your imagination the number of lines it would take in 6502 assembly code!

The program was tested on randomly-ordered lists of length 256 bytes (128 addresses) and 4096 bytes (2048 addresses). Using a stopwatch, the shorter list took 2.0 seconds, and the longer list 35.5 seconds.

A few comments are in order. If the sort routine is to be used by a larger program, parameters (L, R, and the location of the user stack) can be passed by way of page zero. You can actually place them into the appropriate Sweet-16 register from the main program. Another point is that the program, as written, will only work for lists residing in memory locations below $8000. This is because the algorithm used to calculate the middle element of the list is (L + R)/2, and the addition will overflow if L + R exceeds $FFFF. The solution to this problem is to use the fact that L + (R − L)/2 will

Table 1: Sweet-16 Instruction Set

Op-Code	Mnemonic	Operand	Effect
--	@SW16		Enter SWEET16
ØØ	RTN		Return to calling 6502 program
Ø1	BR	(addr)	Unconditional branch
Ø2	BNC	(addr)	Branch if No Carry
Ø3	BC	(addr)	Branch if Carry flag set
Ø4	BP	(addr)	Branch if prior result Plus
Ø5	BM	(addr)	Branch if prior result Minus
Ø6	BZ	(addr)	Branch if prior result Zero
Ø7	BNZ	(addr)	Branch if prior result Not Zero
Ø8	BM1	(addr)	Branch if prior result is -1
Ø9	BNM1	(addr)	Branch if prior result is not -1
ØA	BK		Execute 6502 BRK instruction
ØB	RS		Return from SWEET16 subroutine
ØC	BS	(addr)	Branch to SWEET16 subroutine
ØD			Unassigned
ØE			Unassigned
ØF			Unassigned
1n	SETR	(Rn Constant)	Rn := Constant
2n	LD	(Rn)	RØ := Rn
3n	ST	(Rn)	Rn := RØ
4n	LD@	(Rn)	RØH := Ø; RØL := (Rn); Rn := Rn+1
5n	ST@	(Rn)	(Rn) := RØL; Rn := Rn+1
6n	LDD@	(Rn)	RØL := (Rn); Rn := Rn+1; RØH := (Rn); Rn := Rn+1
7n	STD@	(Rn)	(Rn) := RØL; Rn := Rn+1; (Rn) := RØH; Rn := Rn+1
8n	POP@	(Rn)	Rn := Rn-1; RØL := (Rn); RØH := Ø
9n	STP@	(Rn)	Rn := Rn-1; (Rn) := RØL
An	ADD	(Rn)	RØ := RØ + Rn
Bn	SUB	(Rn)	RØ := RØ - Rn
Cn	POPD@	(Rn)	Rn := Rn-1; RØH := (Rn); Rn := Rn-1; RØL := (Rn)
Dn	CPR	(Rn)	R13 := RØ - Rn; Set status register
En	INCR	(Rn)	Rn := Rn+1 (16-bit increment)
Fn	DECR	(Rn)	Rn := Rn-1 (16-bit decrement)

give the same result without overflow and to modify the program accordingly.

Sweet-16 and the Apple II Plus

When I got my Apple II Plus home from the dealer in February 1980, I was surprised to discover that the miniassembler, the step and trace functions, and Sweet-16 were not included. (The miniassembler and Sweet-16 resided in leftover space in the Integer BASIC ROM and step and trace were casualties of the Autostart ROM.) I soon discovered, however, that the source code for the miniassembler and Sweet-16 were in the "Red Book," the January 1978 edition of the Apple II Reference Manual, which was then provided by Apple with its computers. The source code for Sweet-16 was also published by Steve Wozniak, the author, in the November 1977 issue of *Byte*. I therefore lost little time in relocating first the miniassembler and then Sweet-16 to RAM. I chose to put Sweet-16 at $7689 because I then had a 32K cassette system. (That location also works for a 48K disk system.) The only tricky part of the relocation was the instruction at $F69E/$769E, which had to be changed from LDA #$F7 to LDA #$77. If you were going to relocate Sweet-16 to $9089, for example, this instruction would be changed to LDA #$91. This is the high-order byte used by an internal jump table.

Listing 2 is a disassembled version of the relocated Sweet-16 interpreter that I use in my Apple II Plus.

SPEED16

Sweet-16 spends much of its time decoding instructions. Specifically, it maintains its own program counter (in R15), fetches op-codes, uses a jump table to decide which subroutines to call to execute the op-code, and keeps track of status. All of this takes time, and all of it is overhead.

The Sweet-16 macros discussed above translate the mnemonics into Sweet-16 op-codes, which are then decoded by the Sweet-16 interpreter into the appropriate actions. By taking fuller advantage of the power of the macroassembler, we can eliminate the middleman and translate the mnemonics directly into the appropriate actions. This completely eliminates the need for Sweet-16 to fetch and decode op-codes. In fact, it eliminates the need for op-codes! It also eliminates, as will be seen, the need for a separate program counter and subroutine return stack. With much of the overhead eliminated, the resulting program will run considerably faster.

You may be wondering, "What's the catch?" The catch, if you want to call it that, is that the object code produced will not be nearly as compact as Sweet-16 op-codes. This may or may not be a problem, depending on the application.

All this is accomplished with a new set of macros which I have taken the liberty of calling "SPEED16". The mnemonics and their effects are the same as described in table 1 for Sweet-16, but the method of implementation is quite different. Because the program counter and return stack have been eliminated, the unconditional branch ("BR") and subroutine call ("BS") are implemented using absolute rather than relative addressing. For the same reason, R12 and R15 can be used as general purpose registers. The "@SW16" instruction serves the same purpose as before, but its implementation is greatly simplified; in SPEED16 it merely has to call the monitor's routine for saving registers. Similarly, the "RTN" instruction merely calls the monitor's routine for restoring registers.

The SPEED16 macros are shown in listing 3 with the demonstration program. Only two minor changes were required to the source instructions of the Quicksort routine. Both involved relative branch instructions that had to be changed to absolute branches (because the SPEED16 object code is not as compact) and both were flagged by the assembler. After these two changes were made, the program ran correctly. Note that the object code produced is to the left and below each mnemonic, and is considerably longer than that produced by the Sweet-16 macros. Again using a stopwatch, the progam ran in 1.0 seconds for the 128 element sort and 14.5 seconds for the 2048 element sort.

Some of the SPEED16 macros implement the desired actions directly, including SETR and all the branching instructions. Note that the conditional branching instructions must ascertain the status of the Sweet-16 machine *via* R14H and/or the register containing the last result. The remaining instructions are implemented using calls to subroutines within the Sweet-16 interpreter. Before these subroutines are called, two times the register number must be put in the X register and in R14H. For the CPR instruction, the Y

register must also contain two times 13 (to indicate R13 as the result register). The op-code decoding function of the Sweet-16 interpreter is thus completely bypassed.

Conclusions

Which set of macros should you choose? Using the Quicksort program for the 2048 item sort as a benchmark, some comparisons can be made. Using the Sweet-16 macros, the program assembled in 101 bytes and ran in 35.5 seconds. Using the SPEED16 macros, the program assembled in 586 bytes and ran in 14.5 seconds. It is, thus, a tradeoff of speed *versus* compactness. It should be pointed out, however, that a program can be run under either set of macros with little or no change so that you can use whichever set of macros suits your needs at the moment. Because the instruction set is virtually the same, there is no "learning curve" in switching from one to the other.

Several other comments are in order. First, although the original Sweet-16 has three unused op-codes ($0D, $0E, and $0F), extending the instruction set would be relatively difficult. Because the SPEED16 implementation uses no op-codes, extension of its instruction set using macros is relatively trivial. An example of a desirable extension would be an instruction to move the contents of one register to another without going through the accumulator. Or you might need to add a multiply instruction. Sixteen-bit shift instructions would also be convenient.

One cosmetic change that would simplify the use of either set of macros would be to add "aliases" to several instructions. For example, the STD@ instruction is sometimes used to push a 16-bit quantity onto a stack. An appropriate alias would be the macro

```
!!!PUSHD@  .MD (REG)
           STD@ (REG)
           .ME
```

For 16-bit processing on the 6502, it makes sense to use a virtual machine such as Sweet-16 to simplify the programming effort. Hopefully this article will make the job easier, whether you choose to use the Sweet-16 macros or the SPEED16 macros.

Charles Taylor is a Lieutenant Commander in the U.S. Navy and is currently on the faculty of the Naval Postgraduate School, where he teaches courses in Operations Research, Statistics, and Computer Science. He became involved with computing in 1966, writing ALGOL programs for a Burroughs B5500. He has since worked on and off with computers ranging from the Apple II to the IBM 3033AP and with languages ranging from various assembly languages to Pascal, FORTRAN, C, PILOT, and APL.

Listing 1

```
            ;***** QUICKSORT ****
            ;*                  *
            ;* SWEET16 DEMO     *
            ;*    PROGRAM       *
            ;*                  *
            ;*       BY         *
            ;*                  *
            ;* C.F. TAYLOR,JR.  *
            ;*                  *
            ;*  APRIL 1981      *
            ;*                  *
            ;********************
            ;
            .BA $E000
            .OS
            ;SWEET 16 MACROS
            ;BY R. C. VILE, JR.
            ;MICRO (20:25)
            ;MODIFIED BY
            ;C. F. TAYLOR, JR.
            ;APRIL 1981
!!!SETR     .MD (REG ADDR)
            .BY $10+REG
            .SE ADDR
            .ME
!!!LD       .MD (REG)
            .BY $20+REG
            .ME
!!!ST       .MD (REG)
            .BY $30+REG
            .ME
!!!LD@      .MD (REG)
            .BY $40+REG
            .ME
!!!ST@      .MD (REG)
            .BY $50+REG
            .ME
!!!LDD@     .MD (REG)
            .BY $60+REG
            .ME
!!!STD@     .MD (REG)
            .BY $70+REG
            .ME
!!!POP@     .MD (REG)
            .BY $80+REG
            .ME
!!!STP@     .MD (REG)
            .BY $90+REG
            .ME
!!!ADD      .MD (REG)
            .BY $A0+REG
            .ME
!!!SUB      .MD (REG)
            .BY $B0+REG
            .ME
!!!POPD@    .MD (REG)
            .BY $C0+REG
            .ME
!!!CPR      .MD (REG)
            .BY $D0+REG
            .ME
!!!INCR     .MD (REG)
            .BY $E0+REG
            .ME
!!!DECR     .MD (REG)
            .BY $F0+REG
            .ME
!!!RTN      .MD
            .BY 00
            .ME
!!!RELBR    .MD (LOC)
            .BY LOC-=-1
            .ME
!!!BR       .MD (WHERE)
            .BY 1
            RELBR (WHERE)
            .ME
!!!BNC      .MD (WHERE)
            .BY 2
            RELBR (WHERE)
            .ME
!!!BC       .MD (WHERE)
            .BY 3
            RELBR (WHERE)
            .ME
!!!BP       .MD (WHERE)
            .BY 4
            RELBR (WHERE)
            .ME
!!!BM       .MD (WHERE)
            .BY 5
            RELBR (WHERE)
            .ME
!!!BZ       .MD (WHERE)
            .BY 6
            RELBR (WHERE)
            .ME
!!!BNZ      .MD (WHERE)
            .BY 7
            RELBR (WHERE)
            .ME
!!!BM1      .MD (WHERE)
            .BY 8
            RELBR (WHERE)
            .ME
!!!BNM1     .MD (WHERE)
            .BY 9
            RELBR (WHERE)
            .ME
!!!BK       .MD
            .BY $A
            .ME
!!!RS       .MD
            .BY $B
            .ME
!!!BS       .MD (WHERE)
            .BY $C
            RELBR (WHERE)
            .ME
!!!@SW16    .MD
            JSR $7689
            .ME
;
R0L         .DE $00
R0H         .DE $01
L1          .DE $7500    ;SORT FROM
L2          .DE $75FE    ;TO
I           .DE 1
J           .DE 2
L           .DE 3
R           .DE 4
X           .DE 5
AI          .DE 6
MIDPT       .DE 7
AJ          .DE 8
R10         .DE 10
SP          .DE 11
;
;INITIALIZE
;
            .ES
```

MICRObits

MICRO is increasing the cost of classified insertions (now called MICRObits) to $25.00 each, prepaid, beginning with the January '82 issue. However, MICRO subscribers can advertise in MICRObits, one time, at the old rate of $10.00. (One $10.00 ad per subscriber, subsequent ads $25.00.) To submit an ad to MICRObits, send typewritten copy to reach MICRO by the 20th of the second month before the issue you would like the ad to appear in (i.e., by January 20th for the March issue). 40-word text limit. Late ads held for next issue. Send to:

MICRObits
P.O. Box 6502
Chelmsford, MA 01824

(Continued on page 101)

Listing 1 *(Continued)*

```
                               @SW16          ;CALL SWEET16
6000- 20 89 76
                               SETR (SP $7900)      ;STACK POINTER
6003- 1B
6004- 00 79
                               SETR (R10 $7900)     ;SAVE FOR LATER
6006- 1A
6007- 00 79
                               SETR (0 L1)  ;FIRST ADDR TO SORT
6009- 10
600A- 00 75
                               STD@ (SP)    ;PUSH
600C- 7B
                               SETR (0 L2)  ;LAST ADDR TO SORT
600D- 10
600E- FE 75
                               STD@ (SP)    ;PUSH
6010- 7B
                       ;
                               ;LOOP1 IS THE
                               ;CONTROL LOOP AND
                               ;DECIDES WHICH SUBLIST
                               ;TO PARTITION NEXT
                               ;
                       LOOP1   POPD@ (SP)   ;GET BOUNDARIES
6011- CB
                               ST (R)       ;OF NEXT SUBLIST
6012- 34
                               POPD@ (SP)   ;TO PARTITION
6013- CB
                               ST (L)
6014- 33
                       ;
                               ;LOOP2 DOES THE
                               ;ACTUAL PARTITIONING
                       LOOP2   LD (L)       ;I := L
6015- 23
                               ST (I)
6016- 31
                               LD (R)       ;J := R
6017- 24
                               ST (J)
6018- 32
                               ADD (L)      ;R0 := L+R
6019- A3
                               RTN          ;BACK TO 6502
601A- 00
601B- 46 01                    LSR *R0H     ;DIVIDE BY 2
601D- 66 00                    ROR *R0L
601F- A9 FE                    LDA #$FE     ;MASK ODD BIT
6021- 25 00                    AND *R0L
6023- 85 00                    STA *R0L     ;TO GET EVEN WORD BOUNDARY
                               @SW16        ;BACK TO SWEET16
6025- 20 89 76
                               ST (MIDPT)   ;SAVE RESULT
6028- 37
                               LDD@ (MIDPT)         ;FETCH THAT ITEM
6029- 67
                               ST (X)       ;MIDDLE ELEMENT OF LIST
602A- 35
                               ;
                               ;LOOP3 INTERCHANGES ELEMENTS
                               ;TO ACCOMPLISH PARTITION
                               ;ABOUT MIDPT (VALUE X)
                               ;
                       LOOP3
                               ;
                               ;LOOP4 FINDS AN ELEMENT
                               ;TO LEFT OF MIDPT WITH
                               ;VALUE > X
                               ;
                       LOOP4   LDD@ (I)     ;A(I)
602B- 61
                               ST (AI)      ;SAVE
602C- 36
                               CPR (X)      ;LOOP
602D- D5
                               BNC (LOOP4)  ;UNTIL A(I)>=X
602E- 02
602F- FB
                               DECR (I)     ;ADJUST POINTER
6030- F1
                               DECR (I)
6031- F1
                               ;
```

```
                              ;LOOP5 FINDS AN ELEMENT
                              ;TO RIGHT OF MIDPT WITH
                              ;VALUE < X
                              ;
                   LOOP5      LDD@ (J)       ;A(J)
6032- 62
                              DECR (J)       ;COMPENSATE
6033- F2
                              DECR (J)       ;FOR AUTO-INCR.
6034- F2
                              ST (AJ)
6035- 38
                              LD (X)
6036- 25
                              CPR (AJ)       ;LOOP UNTIL
6037- D8
                              BC (ENDLOOP5)          ;A(J)<=X
6038- 03
6039- 04
                              DECR (J)
603A- F2
                              DECR (J)
603B- F2
                              BR (LOOP5)
603C- 01
603D- F4
                   ENDLOOP5
                              ;
                              LD (J)
603E- 22
                              CPR (I)        ;IF I>J
603F- D1
                          BNC (ENDLOOP3)          ;GOTO ENDLOOP3
6040- 02
6041- 08
                          LD (AI)            ;A(I)
6042- 26
                          STD@ (J)           ;EXCHANGE A(I)
6043- 72
                          DECR (J)           ;AND A(J)
6044- F2
                          DECR (J)           ;ADJ POINTER TO
6045- F2
                          DECR (J)           ;PREVIOUS ELEMENT
6046- F2
                          DECR (J)
6047- F2
                          LD (AJ)            ;A(J)
6048- 28
                          STD@ (I)           ;FINISH EXCHANGE
6049- 71 ENDLOOP3         LD (J)             ;LOOP UNTIL
604A- 22
                          CPR (I)            ;I > J
604B- D1
                          BC (LOOP3)
604C- 03
604D- DD
                          ;
                          LD (R)
604E- 24
                          CPR (I)            ;IF I>=R THEN
604F- D1
                          BNC (ENDIF)        ;GOTO ENDIF
6050- 02
6051- 04
                          LD (I)             ;PUSH FOR LATER
6052- 21
                          STD@ (SP)          ;PARTITIONING
6053- 7B
                          LD (R)             ;(RIGHT PART OF LIST)
6054- 24
                          STD@ (SP)          ;
6055- 7B
                          ;
             ENDIF        LD (J)             ;PREPARE TO PARTITION
6056- 22
                                             ;LEFT PART
                          ST (R)             ;R := J
6057- 34
                          LD (L)
6058- 23
                          CPR (R)            ;IF L<R
6059- D4
                          BNC (LOOP2)
605A- 02
605B- B9
                          LD (R10)           ;ORIG VAL OF SP
605C- 2A
                          CPR (SP)           ;STACK EMPTY?
605D- DB
                          BC (EXIT)          ;YES
605E- 03
605F- 02
                          BR (LOOP1)         ;NO
6060- 01
6061- AF
             EXIT         RTN                ;DONE
6062- 00
6063- 60
                          RTS
                          .EN
```

Listing 2

```
7689-    20 4A FF     JSR    $FF4A
768C-    68           PLA
768D-    85 1E        STA    $1E
768F-    68           PLA
7690-    85 1F        STA    $1F
7692-    20 98 76     JSR    $7698
7695-    4C 92 76     JMP    $7692
7698-    E6 1E        INC    $1E
769A-    D0 02        BNE    $769E
769C-    E6 1F        INC    $1F
769E-    A9 77        LDA    #$77
76A0-    48           PHA
76A1-    A0 00        LDY    #$00
76A3-    B1 1E        LDA    ($1E),Y
76A5-    29 0F        AND    #$0F
76A7-    0A           ASL
76A8-    AA           TAX
76A9-    4A           LSR
76AA-    51 1E        EOR    ($1E),Y
76AC-    F0 0B        BEQ    $76B9
76AE-    86 1D        STX    $1D
76B0-    4A           LSR
76B1-    4A           LSR
76B2-    4A           LSR
76B3-    A8           TAY
76B4-    B9 E1 76     LDA    $76E1,Y
76B7-    48           PHA
76B8-    60           RTS
76B9-    E6 1E        INC    $1E
76BB-    D0 02        BNE    $76BF
76BD-    E6 1F        INC    $1F
76BF-    BD E4 76     LDA    $76E4,X
76C2-    48           PHA
76C3-    A5 1D        LDA    $1D
76C5-    4A           LSR
76C6-    60           RTS
76C7-    68           PLA
76C8-    68           PLA
76C9-    20 3F FF     JSR    $FF3F
76CC-    6C 1E 00     JMP    ($001E)
76CF-    B1 1E        LDA    ($1E),Y
76D1-    95 01        STA    $01,X
76D3-    88           DEY
76D4-    B1 1E        LDA    ($1E),Y
76D6-    95 00        STA    $00,X
76D8-    98           TYA
76D9-    38           SEC
76DA-    65 1E        ADC    $1E
76DC-    85 1E        STA    $1E
76DE-    90 02        BCC    $76E2
76E0-    E6 1F        INC    $1F
76E2-    60           RTS

76E3- 02 F9 04 9D 0D                   ;Jump Table
76E8- 9E 25 AF 16 B2 47 B9 51
76F0- C0 2F C9 5B D2 85 DD 6E
76F8- 05 33 E8 70 93 1E E7 65
7700- E7 E7 E7

7703-    10 CA        BPL    $76CF
7705-    B5 00        LDA    $00,X
7707-    85 00        STA    $00
7709-    B5 01        LDA    $01,X
770B-    85 01        STA    $01
770D-    60           RTS
770E-    A5 00        LDA    $00
```

```
7710-    95 00       STA   $00,X
7712-    A5 01       LDA   $01
7714-    95 01       STA   $01,X
7716-    60          RTS
7717-    A5 00       LDA   $00
7719-    81 00       STA   ($00,X)
771B-    A0 00       LDY   #$00
771D-    84 1D       STY   $1D
771F-    F6 00       INC   $00,X
7721-    D0 02       BNE   $7725
7723-    F6 01       INC   $01,X
7725-    60          RTS
7726-    A1 00       LDA   ($00,X)
7728-    95 00       STA   $00
772A-    A0 00       LDY   #$00
772C-    84 01       STY   $01
772E-    F0 ED       BEQ   $771D
7730-    A0 00       LDY   #$00
7732-    F0 06       BEQ   $773A
7734-    20 66 77    JSR   $7766
7737-    A1 00       LDA   ($00,X)
7739-    A8          TAY
773A-    20 66 77    JSR   $7766
773D-    A1 00       LDA   ($00,X)
773F-    85 00       STA   $00
7741-    84 01       STY   $01
7743-    A0 00       LDY   #$00
7745-    84 1D       STY   $1D
7747-    60          RTS
7748-    20 2E 77    JSR   $7726
774B-    A1 00       LDA   ($00,X)
774D-    85 01       STA   $01
774F-    4C 1F 77    JMP   $771F
7752-    20 17 77    JSR   $7717
7755-    A5 01       LDA   $01
7757-    81 00       STA   ($00,X)
7759-    4C 1F 77    JMP   $771F
775C-    20 66 77    JSR   $7766
775F-    A5 00       LDA   $00
7761-    81 00       STA   ($00,X)
7763-    4C 43 77    JMP   $7743
7766-    B5 00       LDA   $00,X
7768-    D0 02       BNE   $776C
```

```
776A-    D6 01       DEC   $01,X
776C-    D6 00       DEC   $00,X
776E-    60          RTS
776F-    A0 00       LDY   #$00
7771-    38          SEC
7772-    A5 00       LDA   $00
7774-    F5 00       SBC   $00,X
7776-    99 00 00    STA   $0000,Y
7779-    A5 01       LDA   $01
777B-    F5 01       SBC   $01,X
777D-    99 01 00    STA   $0001,Y
7780-    98          TYA
7781-    69 00       ADC   #$00
7783-    85 1D       STA   $1D
7785-    E0          RTS
7786-    A5 00       LDA   $00
7788-    75 00       ADC   $00,X
778A-    85 00       STA   $00
778C-    A5 01       LDA   $01
778E-    75 01       ADC   $01,X
7790-    A0 00       LDY   #$00
7792-    F0 E9       BEQ   $777D
7794-    A5 1E       LDA   $1E
7796-    20 19 77    JSR   $7719
7799-    A5 1F       LDA   $1F
779B-    20 19 77    JSR   $7719
779E-    18          CLC
779F-    B0 0E       BCS   $77AF
77A1-    B1 1E       LDA   ($1E),Y
77A3-    10 01       BPL   $77A6
77A5-    88          DEY
77A6-    65 1E       ADC   $1E
77A8-    85 1E       STA   $1E
77AA-    98          TYA
77AB-    65 1F       ADC   $1F
77AD-    85 1F       STA   $1F
77AF-    60          RTS
77B0-    B0 EC       BCS   $779E
77B2-    60          RTS
77B3-    0A          ASL
77B4-    AA          TAX
77B5-    B5 01       LDA   $01,X
```

```
77B7-    10 E8       BPL   $77A1
77B9-    60          RTS
77BA-    0A          ASL
77BB-    AA          TAX
77BC-    B5 01       LDA   $01,X
77BE-    30 E1       BMI   $77A1
77C0-    60          RTS
77C1-    0A          ASL
77C2-    AA          TAX
77C3-    B5 00       LDA   $00,X
77C5-    15 01       ORA   $01,X
77C7-    F0 D8       BEQ   $77A1
77C9-    60          RTS
77CA-    0A          ASL
77CB-    AA          TAX
77CC-    B5 00       LDA   $00,X
77CE-    15 01       ORA   $01,X
77D0-    D0 CF       BNE   $77A1
77D2-    60          RTS
77D3-    0A          ASL
77D4-    AA          TAX
77D5-    B5 00       LDA   $00,X
77D7-    35 01       AND   $01,X
77D9-    49 FF       EOR   #$FF
77DB-    F0 C4       BEQ   $77A1
77DD-    60          RTS
77DE-    0A          ASL
77DF-    AA          TAX
77E0-    B5 00       LDA   $00,X
77E2-    35 01       AND   $01,X
77E4-    49 FF       EOR   #$FF
77E6-    D0 B9       BNE   $77A1
77E8-    60          RTS
77E9-    A2 18       LDX   #$18
77EB-    20 66 77    JSR   $7766
77EE-    A1 00       LDA   ($00,X)
77F0-    85 1F       STA   $1F
77F2-    20 66 77    JSR   $7766
77F5-    A1 00       LDA   ($00,X)
77F7-    85 1E       STA   $1E
77F9-    60          RTS
77FA-    4C C7 76    JMP   $76C7
```

Listing 3

```
           ;***** QUICKSORT *****
           ;*                   *
           ;* SWEET16 DEMO      *
           ;*   PROGRAM         *
           ;*                   *
           ;*       BY          *
           ;*                   *
           ;* C.F.TAYLOR,JR.    *
           ;*                   *
           ;*  APRIL 1981       *
           ;*                   *
           ;*********************
           ;
           .BA $6000
           .OS
;
L1         .DE $7500      ;SORT FROM HERE
L2         .DE $75FE      ;TO HERE
I          .DE 1
J          .DE 2
L          .DE 3
R          .DE 4
X          .DE 5
AI         .DE 6
MIDPT      .DE 7
AJ         .DE 8
R10        .DE 10
SP         .DE 11
;
           ;**** SPEED16 MACROS ****
           ;*                       *
           ;*          BY           *
           ;*                       *
           ;*    C.F. TAYLOR,JR.    *
           ;*                       *
           ;*      APRIL 1981       *
           ;*                       *
           ;*************************
           ;
           ;SWEET16 SUBROUTINES
LOAD       .DE $7705
STORE      .DE $770E
LOADI      .DE $7726
STOREI     .DE $7717
DLOADI     .DE $7748
DSTOREI    .DE $7752
POPS       .DE $7730
PUSHS      .DE $775C
ADDITION   .DE $7786
SUBTRACT   .DE $776F
POP16      .DE $7734
COMPARE    .DE $7771
INCREMENT  .DE $771F
DECREMENT  .DE $7766
           ;
           ;PAGE ZERO LOCATIONS
R0L        .DE $00
R0H        .DE $01
R14H       .DE $1D
           ;
           ;MACRO DEFINITIONS
           ;
!!!SETUP   .MD (REG)       ;FOR SUBR CALL
           LDA #REG
           ASL A           ;DOUBLE TO INDEX
           STA *R14H       ;16-BIT REG
           TAX
           .ME
;
!!!SETR    .MD (REG ADDR)
           SETUP (REG)
           LDA #H,ADDR     ;HI BYTE
           STA *R0H,X      ;IN PROPER REG
           LDA #L,ADDR     ;LO BYTE
           STA *R0L,X
           .ME
!!!LD      .MD (REG)
           SETUP (REG)
           JSR LOAD
           .ME
!!!ST      .MD (REG)
           SETUP (REG)
           JSR STORE
```

```
         .ME
!!!LD@   .MD (REG)
         SETUP (REG)
         JSR LOADI
         .ME
!!!ST@   .MD (REG)
         SETUP (REG)
         JSR STOREI
         .ME
!!!LDD@  .MD (REG)
         SETUP (REG)
         JSR DLOADI
         .ME
!!!STD@  .MD (REG)
         SETUP (REG)
         JSR DSTOREI
         .ME
!!!POP@  .MD (REG)
         SETUP (REG)
         JSR POPS
         .ME
!!!STP@  .MD (REG)
         SETUP (REG)
         JSR PUSHB
         .ME
!!!ADD   .MD (REG)
         SETUP (REG)
         JSR ADDITION
         .ME
!!!SUB   .MD (REG)
         SETUP (REG)
         JSR SUBTRACT
         .ME
!!!POPD@ .MD (REG)
         SETUP (REG)
         JSR POP16
         .ME
!!!CPR   .MD (REG)
         SETUP (REG)
         JSR COMPARE
         .ME
!!!INCR  .MD (REG)
         SETUP (REG)
         JSR INCREMENT


         .ME
!!!DECR  .MD (REG)
         SETUP (REG)
         JSR DECREMENT
         .ME
!!!RTN   .MD                ;TO 6502 CODE
         JSR $FF3F          ;RESTORE REGS
         .ME
!!!GETCARR .MD
         LDA *R14H          ;STATUS REG
         LSR A              ;EXTRACT CARRY BIT
         .ME
!!!BR    .MD (DEST)
         JMP DEST
         .ME
!!!BNC   .MD (DEST)
         GETCARR
         BCC DEST
         .ME
!!!BC    .MD (DEST)
         GETCARR
         BCS DEST
         .ME
!!!TEST  .MD                ;GET PREV RESULT
         GETCARR
         ASL A              ;GET PRIOR REG #
         TAX
         .ME
!!!BP    .MD (DEST)
         TEST
         LDA *R0H,X
         BPL DEST
         .ME
!!!BM    .MD (DEST)
         TEST
         LDA *R0H,X
         BMI DEST
         .ME
!!!BZ    .MD (DEST)
         TEST
```



```
         LDA *R0L,X         ;TEST FOR 0
         ORA *R0H,X         ;(BOTH BYTES)
         BEQ DEST
         .ME
!!!BNZ   .MD (DEST)
         TEST
         LDA *R0L,X         ;TEST BOTH
         ORA *R0H,X         ;BYTES
         BNE DEST
         .ME
!!!BM1   .MD (DEST)
         TEST
         LDA *R0L,X         ;TEST FOR $FF=-1
         AND *R0H,X         ;(BOTH BYTES)
         EOR #$FF
         BEQ DEST
         .ME
!!!BNM1  .MD (DEST)
         TEST
         LDA *R0L,X
         AND *R0H,X
         EOR #$FF
         BNE DEST
         .ME
!!!BK    .MD
         BRK                ;6502 INSTR
         .ME
!!!RS    .MD
         RTS                ;6502 INSTR
         .ME
!!!BS    .MD (DEST)
         JSR DEST           ;DIRECT ADDR MODE
         .ME
!!!@SW16 .MD
         JSR $FF4A          ;SAVE REGISTERS
         .ME
;
         .LS
;INITIALIZE
;
         .ES
         @SW16              ;CALL SWEET16
```

```
6000- 20 4A FF  SETR (SP $7900)        ;STACK POINTER

6003- A9 0B
6005- 0A
6006- 85 1D
6008- AA
6009- A9 79
600B- 95 01
600D- A9 00
600F- 95 00      SETR (R10 $7900)      ;SAVE FOR LATER

6011- A9 0A
6013- 0A
6014- 85 1D
6016- AA
6017- A9 79
6019- 95 01
601B- A9 00
601D- 95 00      SETR (0 L1)  ;FIRST ADDR TO SORT

601F- A9 00
6021- 0A
6022- 85 1D
6024- AA
6025- A9 75
6027- 95 01
6029- A9 00
602B- 95 00      STD@ (SP)     ;PUSH

602D- A9 0B
602F- 0A
6030- 85 1D
6032- AA
6033- 20 52 77 SETR (0 L2)  ;LAST ADDR TO SORT

6036- A9 00
6038- 0A
6039- 85 1D
603B- AA
603C- A9 75
603E- 95 01
6040- A9 FE
```

```
6042- 95 00
                 STD@ (SP)   ;PUSH
6044- A9 0B
6046- 0A
6047- 85 1D
6049- AA
604A- 20 52 77
                 ;
                             ;LOOP1 IS THE
                             ;CONTROL LOOP AND
                             ;DECIDES WHICH SUBLIST
                             ;TO PARTITION NEXT
                             ;
                LOOP1        POPD@ (SP)    ;GET BOUNDARIES
604D- A9 0B
604F- 0A
6050- 85 1D
6052- AA
6053- 20 34 77
                             ST (R)        ;OF NEXT SUBLIST

6056- A9 04
6058- 0A
6059- 85 1D
605B- AA
605C- 20 0E 77
                             POPD@ (SP)    ;TO PARTITION

605F- A9 0B
6061- 0A
6062- 85 1D
6064- AA
6065- 20 34 77
                             ST (L)

6068- A9 03
606A- 0A
606B- 85 1D
606D- AA
606E- 20 0E 77
                 ;
                             ;LOOP2 DOES THE
                             ;ACTUAL PARTITIONING
                LOOP2        LD (L)        ;I := L
6071- A9 03
6073- 0A
```

```
6074- 85 1D
6076- AA
6077- 20 05 77             ST (I)

607A- A9 01
607C- 0A
607D- 85 1D
607F- AA
6080- 20 0E 77             LO (R)       ;J := R

6083- A9 04
6085- 0A
6086- 85 1D
6088- AA
6089- 20 05 77             ST (J)

608C- A9 02
608E- 0A
608F- 85 1D
6091- AA
6092- 20 0E 77             ADD (L)      ;R0 := L+R

6095- A9 03
6097- 0A
6098- 85 1D
609A- AA
609B- 20 86 77             RTN          ;BACK TO 6502

609E- 20 3F FF
60A1- 46 01                LSR *R0H     ;DIVIDE BY 2
60A3- 66 00                ROR *R0L
60A5- A9 FE                LDA #$FE     ;MASK ODD BIT
60A7- 25 00                AND *R0L
60A9- 85 00                STA *R0L     ;TO GET EVEN WORD BOUNDARY
                           @SW16        ;BACK TO SWEET16

60AB- 20 4A FF             ST (MIDPT)   ;SAVE RESULT

60AE- A9 07
60B0- 0A
60B1- 85 1D
60B3- AA
60B4- 20 0E 77             LDD@ (MIDPT)       ;FETCH THAT ITEM

60B7- A9 07
60B9- 0A
60BA- 85 1D
60BC- AA
60BD- 20 48 77             ST (X)       ;MIDDLE ELEMENT OF LIST

60C0- A9 05
60C2- 0A
60C3- 85 1D
60C5- AA
60C6- 20 0E 77
                           ;
                           ;LOOP3 INTERCHANGES ELEMENTS
                           ;TO ACCOMPLISH PARTITION
                           ;ABOUT MIDPT (VALUE X)
                           ;
                   LOOP3
                           ;
                           ;LOOP4 FINDS AN ELEMENT
                           ;TO LEFT OF MIDPT WITH
                           ;VALUE ) X
                           ;
                   LOOP4   LDD@ (I)     ;A(I)
60C9- A9 01
60CB- 0A
60CC- 85 1D
60CE- AA
60CF- 20 48 77             ST (AI)      ;SAVE

60D2- A9 06
60D4- 0A
60D5- 85 1D
60D7- AA
60D8- 20 0E 77             CPR (X)      ;LOOP

60DB- A9 05
60DD- 0A
60DE- 85 1D
60E0- AA
60E1- 20 71 77             BNC (LOOP4)  ;UNTIL A(I))=X

60E4- A5 1D
60E6- 4A
60E7- 90 E0                DECR (I)     ;ADJUST POINTER
```

Lazer isn't afraid to compare!

(because we have the best lower case system available.)

Despite the fact that we were one of the first manufacturers to produce lower case equipment for the Apple II, Lazer MicroSystems products are still the state-of-the-art. Beside the obvious price/performance advantage we have over the competition, our products are expandable. Lazer is constantly introducting new products including our Lower Case + Plus II, Character Set + Plus (that adds 2 additional character sets to the Lower Case + Plus), and our new "Double Vision + Plus" for owners of Computer STOP's Double Vision 80-column board.

Lazer's products are compatable with more word processors than anybody else's. Our Lower Case + Plus is compatable with Easywriter and unmodified Apple writers. None of our competitors below can make that claim. In fact, BASIS' board isn't even compatable with Programma's PIE! The following chart lists Lazer's superiority over the competing units.

[LC+ = Lower Case + Plus, LC+II = Lower Case + Plus II, KB+ = Keyboard + Plus]

Feature	Paymar LCA-1	Paymar LCA-2	VIDEX	BASIS	VISTA	LC+	LC+II	KB+/ LC+II	KB+/ LC+	KB+
True ASCII upper/lower case display	Y	Y	Y	Y	N	Y	Y	Y	Y	N
Inverse Lower Case	N	N	rev 7 only	N	—	Y	N	N	Y	—
Font Size	5 x 7	5 x 7	5 x 8	5 x 8	—	5x7, 7x8	5 x 7	5 x 7	5x7, 7x8	—
# of on-board character sets	1	1	1	1	—	up to 4 (2 std)	1	1	up to 4	—
Pseudo-descenders	Y	Y	N	N	—	Y	Y	Y	Y	—
True descenders	N	N	Y	Y	—	optional	N	N	optional	—
Optional fonts avail. (ROM, disk)	N	N	N	Y	—	Y	N	N	Y	—
2716-compatible character generator compatable with fonts created by HIRES character generators	N	N	N	N	—	Y	N	N	Y	—
On-board graphics character set	N	N	N	N	—	Y	N	N	Y	—
Software provided on diskette	$5 extra		N	N	—	Y	N	N	Y	Y
Single board works with all Apples	N	N	N	N	Y	Y	N	N	Y	Y
Expandable System	N	N	N	N	N	Y	Y	Y	Y	Y
Extensive user Documentation	N	N	Y	N	N	Y	Y	Y	Y	Y
High quality PC board	N	—	Y	Y	Y	Y	—	Y	Y	Y
Reset key disable	N	N	Y	Y	N	N	N	Y	Y	Y
Shift key mod	N	N	Y	Y	N	N	N	Y	Y	Y
All 128 characters available from keyboard	—	—	N	N	—	—	—	Y	Y	Y
Type ahead buffer	N	N	N	N	Y	N	N	Y	Y	Y
# of characters in buffer	—	—	—	—	40	—	—	64	64	64
Ability to clear or turn off buffer	—	—	—	—	N	—	—	Y	Y	Y
PRICE	59.95	49.95	129.95	125.00	49.95	64.95	29.95	129.90	164.90	99.95

Don't settle for anything less than the Lower Case + Plus, Lower Case + Plus II, or Keyboard + Plus. If your dealer doesn't stock our products give us a call, we'll try to connect you with a dealer in your area.

Before you buy a lower case system for the Apple II, call or write for our free booklet "Keyboard Enhances/Buffers and Lower Case Adapters: From The Inside Out." This booklet explains the advantages and disadvantages of using a lower case adapter in your Apple II.

Lazer products are carried by reputable dealers all across the world. If your dealer cannot provide you with a demonstration of our equipment and tries to sell you an inferior lower case adapter, give us a call, we'll give you the location of a dealer that can show you our equipment. Remember, an intelligent purchase cannot be made if you do not compare before buying.

State of the art performance, software compatability, and exceptional value make Lazer's products the best there are!

Lower Case + Plus $64.95
Keyboard + Plus $99.95
Lower Case + Plus II $29.95
Character Set + Plus $24.95
Double Vision + Plus $39.95

Anix 1.0 .... $49.95
Lazer Pascal $39.95

Visicalc Users! Now you can have lower case on your Visicalc Screen, *only* from Lazer.

Lazer MICRO SYSTEMS INC.

1791-G Capital
Corona, CA 91720
(714) 735-1041

```
60E9- A9 01
60EB- 0A
60EC- 85 1D
60EE- AA
60EF- 20 66 77
                                  DECR (I)

60F2- A9 01
60F4- 0A
60F5- 85 1D
60F7- AA
60F8- 20 66 77
                                  ;
                                  ;LOOP5 FINDS AN ELEMENT
                                  ;TO RIGHT OF MIDPT WITH
                                  ;VALUE < X
                                  ;
                        LOOP5     LODB (J)       ;A(J)
60FB- A9 02
60FD- 0A
60FE- 85 1D
6100- AA
6101- 20 48 77
                                  DECR (J)       ;COMPENSATE
6104- A9 02
6106- 0A
6107- 85 1D
6109- AA
610A- 20 66 77
                                  DECR (J)       ;FOR AUTO-INCR.
610D- A9 02
610F- 0A
6110- 85 1D
6112- AA
6113- 20 66 77
                                  ST (AJ)
6116- A9 08
6118- 0A
6119- 85 1D
611B- AA
611C- 20 0E 77
                                  LD (X)
611F- A9 05
6121- 0A
6122- 85 1D
6124- AA
6125- 20 05 77
                                  CPR (AJ)       ;LOOP UNTIL
6128- A9 08
612A- 0A
612B- 85 1D
612D- AA
612E- 20 71 77
                                  BC (ENDLOOP5)        ;A(J)<=X
6131- A5 1D
6133- 4A
6134- B0 15
                                  DECR (J)
6136- A9 02
6138- 0A
6139- 85 1D
613B- AA
613C- 20 66 77
                                  DECR (J)
613F- A9 02
6141- 0A
6142- 85 1D
6144- AA
6145- 20 66 77
                                  BR (LOOP5)
6148- 4C FB 60
                        ENDLOOP5
                                  ;
                                  LD (J)
614B- A9 02
614D- 0A
614E- 85 1D
6150- AA
6151- 20 05 77
                                  CPR (I)        ;IF I>J
6154- A9 01
6156- 0A
6157- 85 1D
6159- AA
615A- 20 71 77
                                  BNC (END3)     ;GOTO END3
615D- A5 1D
```

```
615F- 4A
6160- 90 48
                        LD (A1)       ;A(1)
6162- A9 06
6164- 0A
6165- 65 1D
6167- AA
6168- 20 05 77
                        STD@ (J)      ;EXCHANGE A(I)
616B- A9 02
616D- 0A
616E- 85 1D
6170- AA
6171- 20 52 77
                        DECR (J)      ;AND A(J)
6174- A9 02
6176- 0A
6177- 85 1D
6179- AA
617A- 20 66 77
                        DECR (J)      ;ADJ POINTER TO
617D- A9 02
617F- 0A
6180- 85 1D
6182- AA
6183- 20 66 77
                        DECR (J)      ;PREVIOUS ELEMENT
6186- A9 02
6188- 0A
6189- 85 1D
618B- AA
618C- 20 66 77
                        DECR (J)
618F- A9 02
6191- 0A
6192- 85 1D
6194- AA
6195- 20 66 77
                        LD (AJ)       ;A(J)
6198- A9 08
619A- 0A
619B- 85 1D
619D- AA
619E- 20 05 77
                        STD@ (I)      ;FINISH EXCHANGE
61A1- A9 01
61A3- 0A
61A4- 85 1D
61A6- AA
61A7- 20 52 77
               END3     LD (J)        ;LOOP UNTIL
61AA- A9 02
61AC- 0A
61AD- 85 1D
61AF- AA
61B0- 20 05 77
                        CPR (1)       ;1 ) J
61B3- A9 01
61B5- 0A
61B6- 85 1D
61B8- AA
61B9- 20 71 77
                        BNC (ENDLOOP3)
61BC- A5 1D
61BE- 4A
61BF- 90 03
                        BR (LOOP3)
61C1- 4C C9 60
               ENDLOOP3
                        ;
                        LD (R)
61C4- A9 04
61C6- 0A
61C7- 85 1D
61C9- AA
61CA- 20 05 77
                        CPR (1)       ;IF 1)=R THEN
61CD- A9 01
61CF- 0A
61D0- 85 1D
61D2- AA
61D3- 20 71 77
                        BNC (ENDIF)   ;GOTO ENDIF
61D6- A5 1D
61D8- 4A
```

```
61D9- 90 24
                        LD (I)        ;PUSH FOR LATER
61DB- A9 01
61DD- 0A
61DE- 85 1D
61E0- AA
61E1- 20 05 77
                        STD@ (SP)     ;PARTITIONING
61E4- A9 0B
61E6- 0A
61E7- 85 1D
61E9- AA
61EA- 20 52 77
                        LD (R)        ;(RIGHT PART OF
                                      ; LIST)
61ED- A9 04
61EF- 0A
61F0- 85 1D
61F2- AA
61F3- 20 05 77
                        STD@ (SP)
61F6- A9 0B
61F8- 0A
61F9- 85 1D
61FB- AA
61FC- 20 52 77
                        ;
               ENDIF    LD (J)        ;PREPARE TO
                                      ; PARTITION
61FF- A9 02
6201- 0A
6202- 85 1D
6204- AA
6205- 20 05 77
                                      ;LEFT PART
                        ST (R)        ;R := J
6208- A9 04
620A- 0A
620B- 85 1D
620D- AA
620E- 20 0E 77
                        LD (L)
6211- A9 03
6213- 0A
6214- 85 1D
6216- AA
6217- 20 05 77
                        CPR (R)       ;IF L<R
621A- A9 04
621C- 0A
621D- 85 1D
621F- AA
6220- 20 71 77
                        BC (ENDLOOP2)
6223- A5 1D
6225- 4A
6226- B0 03
                        BR (LOOP2)
6228- 4C 71 60
               ENDLOOP2
                        LD (R10)      ;ORIG VAL OF SP
622B- A9 0A
622D- 0A
622E- 85 1D
6230- AA
6231- 20 05 77
                        CPR (SP)      ;STACK EMPTY?
6234- A9 0B
6236- 0A
6237- 85 1D
6239- AA
623A- 20 71 77
                        BC (EXIT)     ;YES
623D- A5 1D
623F- 4A
6240- B0 03
                        BR (LOOP1)    ;NO
6242- 4C 4D 60
               EXIT     RTN           ;DONE
6245- 20 3F FF
6248- 60
                        .RTS
                        .EN
```

MONEY BUSINESS

Beat the I.R.S.

You'd always pay the minimum Federal income tax if you had the knowledge and time to examine 120 tax return options. Well, now you can do it, with Datamost's TAX BEATER.

Written by an I.R.S. Enrolled Agent, with 24 years of financial and tax planning experience, the TAX BEATER automatically evaluates up to 120 return options from your input. It searches and finds the best tax path for you . . . displaying up to 15, and ready to print out the optimum method so you pay the minimum!

The TAX BEATER can handle it all. From income averaging to loss carryovers to dividends, alimony, pension to special exclusions. It's perfect for the average taxpayer . . . indispensable for the financial professional. It's so complete, so logical that, especially in this interim year of unusual tax revisions, it can easily pay for itself many times over. Be ready for the I.R.S. . . . with TAX BEATER, the program that's updated for the 1981 tax laws!

$129.95 on disk for Apple II*. Complete with thorough, easy to follow documentation.

Beat the Real Estate game

Real Estate has always been the major wealth-builder. In fact, more millionaire dynasties owe their creation to investments in real estate than anything else. And, it doesn't matter . . . boom times or bust . . . inflation or deflation . . . the knowing investor can make money, even create an empire!

But, the key to real estate success is accurate analysis of opportunities, knowledge of cash flows, return on investment, current and future profitability, tax consequences and other interrelated information — all needed, all necessary to making the right decision.

And that's what REAP (Real Estate Analysis Program) is all about. It takes your input, probes for the right information, scrutinizes the data, the opportunities . . . and automatically delivers the facts . . . showing you the possibilities and why one situation may be superior to another. In short, REAP can help you reap bigger profits. That's why REAP is the program for any investor . . . a must program for the sophisticated investor. Don't option, commit, invest or buy without it . . . whether you just want to make good money or create a successful real estate empire!

$129.95 on disk for Apple II*. Complete with thorough easy to follow documentation.

VISA/MASTERCHARGE accepted.
$1.00 shipping/handling charge.
(California residents add 6% tax)

Available at computer stores, or from:

DATAMOST
19273 Kenya St.
Northridge, Ca. 91326
(213) 366-7160

* Apple II is a trademark of Apple Computer, Inc

Epson MX80 Interface for SYM-1

This article describes tha hardware and sottware needed for a parallel interfaca of an Epson MX-80 printar to a SYM-1. Tha software Illustrates the use of the SYM EXECUTE and USER commands. The interface also sarves as an example application of the 6522 VIA.

Richard H. Turpin
8226 Warbler Way
Indianapolis, Indiana 46256

When I bought my Epson MX-80, I decided not to order an interface cable, but to make my own since commercially available cables are quite expensive. The MX-80 standard interface is a parallel interface, with a serial interface offered as an option.

The parallel interface requires an Amphenol connector, type number 57-10360 or equivalent, which I purchased at a local electronics store. After studying the printer manual, I decided to interface the printer to the SYM auxiliary applications (AA) connector, and use tbe 6522 VIA #2 to provide the required I/O.

Figure 1 gives the interconnections. Port A is the character output port, with control line CA2 configured in the pulse mode to strobe data into the printer buffer after it has heen written to the port. CA1 senses the printer ready line. Port B provides further I/O such as sensing printer status and selecting/deselecting the printer.

The code I wrote to drive the MX-80 was designed to provide convenient control of the printer operation from the computer terminal. USER commands are implemented to turn on and off tbe compressed character, douhle strike, and emphasized printer modes. It is also possible to specify the line spacing. All the control codes use USR 0. To turn on the compressed character format, for example, you would type U0 1 ↵ (↵ denotes carriage return) at the terminal. To set up douhle spacing type U0 9-18 ↵ .

A listing of the interface code is given in figure 2. New and potential MX-80 owners and SYM-1 owners should be interested in the example applications of the SYM EXECUTE and USR commands. The EXECUTE command is used to initialize the 6522 VIA and to attaeh the printer driver code to the SYM-1 monitor. To use the code given here, simply type E 7000 .

Figure 1: **SYM-1/MX-80 Interface**

Ribbon Cable	MX-80 Connector	Signal	SYM AA Connector (VIA 2)	
1	1	STROBE	AA-4, CA2 (pulse mode)	
2	2	D1	AA-D	
3	3	D2	AA-3	
4	4	D3	AA-C	
5	5	D4	AA-12	Port A
6	6	D5	AA-N	
7	7	D6	AA-11	
8	8	D7	AA-M	
9	9	D8	AA-10	
10	10	ACKNLG	AA-E, CA1	
11	11	BUSY	AA-6, PB7	
12	12	PE	AA-H, PB6	
13	13	SLCT	AA-7, PB5	
14	14	AUTO FEED XT	AA-9, PB1	
15	16	Logic Gnd	AA-1, Gnd	
16	31	INIT	AA-L, PB0	
17	36	SLCT IN	AA-K, PB2	
18	32	ERROR	AA-J, PB4	
19	16	Logic Gnd	AA-1, Gnd	
20	16	Logic Gnd	AA-1, Gnd	

(Pins 19 through 30 connected to pin 16 on MX-80 connector.)

Figure 2

```
                        ;
                        ; Program to interface MX80 printer to SYM
                        ; by Richard H. Turpin, April 11, 1981
                        ;
                                  .ba $7000
                        ;
                        ; Initialize the VIA
7000- 47 37 30                    .by 'G701F' $D
7003- 31 46 0D
                        ;
                        ; Attach the code to SYM monitor.
7006- 73 64 37                    .by 'sd7037,A664' $D
7009- 30 33 37
700C- 2C 41 36
700F- 36 34 0D
                        ;
                        ; Attach USR0 code to monitor
7012- 53 44 37                    .by 'SD7055,A66D' $D 0
7015- 30 35 35
7018- 2C 41 36
701B- 36 44 0D
701E- 00
                        ;
                        ;
                        ; Define external references
                        crt       .de $8aa0    SYM RS-232 service routine
                        lstcom    .de $A657
                        via       .de $a800
                        portb     .de via
                        porta     .de via+1
                        ddrb      .de via+2
                        ddra      .de via+3
                        pcr       .de via+$c
                        ifr       .de via+$d
                        ;
                        ; Initialize VIA
701F- A9 FF             INIT      lda #$ff     data output (port A)
7021- 8D 03 A8                    sta ddra
7024- A9 01                       lda #1
7026- 8D 02 A8                    sta ddrb     INIT control bit 0
7029- 8D 00 A8                    sta portb
702C- A9 0A                       lda #$a
702E- 8D 0C A8                    sta pcr      CA2 pulse mode, CA1 neg. edge sense
7031- A9 00                       lda #0
7033- 8D 01 A8                    sta porta    dummy write to arm flag (IFR)
7036- 60                          rts
                        ;
                        ; Printing code to attach to OUTCHR
                        ; Test for printer on line..if not, simply display on CRT
7037- 48                print     pha          save character
7038- A9 20                       lda #$20
703A- 2C 00 A8                    bit portb    check printer on line
703D- F0 05                       beq skipit   (0 if off line)
703F- 68                          pla          retrieve character
7040- 20 48 70                    jsr toMX80   print it
7043- 48                          pha
7044- 68                skipit    pla
7045- 4C A0 8A                    jmp crt now to the CRT service routine
                        ;
                        ; Subroutine to output character to MX80
7048- 48                toMX80    pha          save character
7049- A9 02                       lda #2
704B- 2C 0D A8                    bit ifr      wait MX80 ready
704E- F0 FB                       beq =-4
7050- 68                          pla          retrieve character
7051- 8D 01 A8                    sta porta    write it
7054- 60                          rts
                        ;
                        ; USR0 code to control MX80 format.
                        ;
                        ;  code no.        function
                        ;     1        compressed character ON
                        ;     2        compressed character OFF
                        ;     3        double strike ON
                        ;     4        double strike OFF
                        ;     5        emphasized ON
                        ;     6        emphasized OFF
                        ;     7        select printer
                        ;     8        deselect printer
                        ;     9        set line spacing (parameter 2 = spacing,
                        ;                 $C for normal, $18 for double spacing)
                        ;
                        ; USR parameter 3 storage
                        p31       .de $A64A
                        ;
7055- CD 57 A6          prcon     cmp lstcom   test valid entry
7058- F0 02                       beq p.of
```

```
705A- 38            p.bad      sec            flag illegal activity
705B- 60                       rts
705C- C9 14         p.of       cmp #$14       is it USR0?
705E- D0 FA                    bne p.bad      no..
7060- E0 03                    cpx #3         3 params not allowed
7062- F0 F6                    beq p.bad
7064- E0 02                    cpx #2         2 params for line spacing
7066- F0 30                    beq p.line
7068- AD 4A A6                 lda p31 (1 parameter) get parameter
706B- 29 0F                    and #$f
706D- C9 09                    cmp #$9 can't be greater than 8
706F- 10 E9                    bpl p.bad
7071- AA                       tax            move to index
7072- BD B6 70                 lda table,x    lookup control character
7075- 10 09                    bpl p.skip
7077- 29 7F                    and #$7f       mask off msb (ESC char. flag)
7079- 48                       pha            save character
707A- A9 1B                    lda #$1b send ESC character
707C- 20 48 70                 jsr toMX80
707F- 68                       pla
7080- C9 11         p.skip     cmp #$11 is it printer select?
7082- F0 0D                    beq p.sel
7084- C9 13                    cmp #$13 is it printer deselect?
7086- F0 05                    beq p.des
7088- 20 48 70                 jsr toMX80   write control character
708B- 18                       clc
708C- 60                       rts
708D- A9 00         p.des      lda #0 deselect printer via port B, bit 0
708F- F0 02                    beq p.sel+2
7091- A9 01         p.sel      lda #1 select printer
7093- 8D 00 A8                 sta portb
7096- 18                       clc
7097- 60                       rts
                    ;
                    ; Routine to set line spacing.
                    ; Second parameter specifies new spacing.
                    ; Hex C is standard, hex 18 gives double spacing.
7098- A9 1B         p.line     lda #$1b       output ESC character
709A- 20 48 70                 jsr toMX80
709D- A9 41                    lda #'A          followed by an A
709F- 20 48 70                 jsr toMX80
70A2- AD 4A A6                 lda p31 get new line spacing
70A5- 09 80                    ora #$80       add 128
70A7- 20 48 70                 jsr toMX80
70AA- A9 1B                    lda #$1b         send another ESC
70AC- 20 48 70                 jsr toMX80
70AF- A9 32                    lda #'2          followed by a 2
70B1- 20 48 70                 jsr toMX80
70B4- 18                       clc
70B5- 60                       rts
                    ; Table of MX80 command characters
70B6- 00 0F 12      table      .by 0 $0F $12 $c7 $c8 $c5 $c6 $11 $13
70B9- C7 C8 C5
70BC- C6 11 13
```

MICRO

Letterbox *(Continued from page 6)*

What About Atari?

Dear Editor:

I would like to praise the inclusion of the new "From Here to Atari" column by James Capparell. I feel the Atari is a fine machine and deserves more recognition. Hopefully, the new column signals a growing interest on your part to cover the Atari computer more fully.

The Atari seems to have gained a reputation as a game computer, and while the games are of course good, the reason I bought an 800 instead of a VCS was that I wanted it to do more. I hope we can look forward to some practical applications appearing in MICRO soon.

Perhaps the continuation of game-only articles becomes a self-fulfilling prophecy, and perpetuates an undeserved reputation.

In the future I, and I'm sure other Atari owners, would appreciate your increased attention to serious Atari applications.

Michael B. Moore
San Francisco, CA

Editor's Note: We encourage our readers who use Ataris to send in articles and applications they've developed. We are aware of the growing popularity of the Atari and want to expand our coverage of it, but need your help.

A Call for Chemistry Programs

Dear Editor:

My science education class acquired an Apple II Plus this Fall. Now my chemistry students practice writing formulas and equations, they study the periodic table and account for energy changes between atoms, they review for tests and check their knowledge with immediate confirmation, and they learn the ways of atoms and electrons through animation and simulations — on the computer!

I'd like to find out about programs for drills, graphics, and simulations — anything useful for chemistry class. Perhaps other teachers could help.

Judy Toop
Auburn Adventist Academy
Auburn, Washington 98002
(206) 939-5000, Ext. 236

MICRO


"THIEF"
"The damn things nearly killed me."
THIEF
It seemed like a cinch assignment. At least for the galaxy's master thief.
Just sneak inside the Star-Complex Base, dodge a few guardian robots and grab the formula. A piece of cake.
Oh yeah! They didn't say the robots went berserk at the sight of a human. They didn't say I'd have to laser my way through 7 levels. Or, that rampaging robots would keep coming and coming.
And most of all, they didn't bother to say there's some weird blob, an unearthly something that tracks you right thru the damn walls.
I made it to the 7th level. I found the formula. But, more robots are massing out there. And I'm wounded. My energy charge is low.
You. Yes, you. How good a Thief are you? Prove yourself. Come get me. Come save me.
$29.95 for Apple II*. At computer stores, or from DATAMOST.
VISA/MASTERCHARGE accepted. $1.00 shipping/handling charge. (CA residents add 6% tax)
*Apple II is a trademark of Apple Computer, Inc.
DATAMOST
19273 Kenya St., Northridge, CA 91326, (213) 366-7160

MICRO

PET Vet

By Loren Wright

What is Pascal?

Pascal was conceived by Niklaus Wirth in 1970-71 as a "complete" language, capable of the most complex structures, yet still easy to learn and program. Although the language was fairly well-specified by Wirth, there was room for improvement. Several different versions have been written, all consistent with the basic structure of the language, but different in several minor respects.

Probably the most popular implementation is called UCSD Pascal, which was developed at the University of California at San Diego. UCSD Pascal adds some important enhancements, particularly a convenient method of handling character strings, which make it very attractive for microcomputer implementations.

Another version, called "Tiny" Pascal, was described by Yuen and Chung in the September-November 1978 issues of *BYTE*. As its name implies, there is quite a bit missing, including some of the more powerful features of the original. Nevertheless, it does make the language available where memory is severely restricted (such as in a 16K PET).

To learn more about Pascal, read Victor Fricke's series, which began in our November issue and concludes in this issue. He cites a number of other books.

Pascal is generally a compiled language. However, microcomputer implementations ordinarily use a compiler which produces P-code. A P-code interpreter is then required for execution. The typical microcomputer Pascal package includes three programs: an editor, a compiler, and a P-code interpreter. As we shall see, a couple of the PET packages are not typical.

Tiny Pascal — Abacus Software

Tiny Pascal, with some enhancements, is intended primarily for learning the language. There are actually two versions. One, called just Tiny Pascal, runs in 16K. The other, called Tiny Pascal Plus+, requires 32K, but includes enhancements for graphics support (limited to the quarter-boxes for 80 × 50 resolution plotting).

The editor is a traditional line-oriented editor, where lines may be inserted, deleted, moved, searched, listed, etc. The familiar screen-editing capabilities of the PET itself are not supported. When a source file is saved, all but the last line is saved. (This problem may not be present in all versions, but I have called it to the attention of Abacus and they intend to correct it.)

The compiler is written in BASIC, which means it is very slow. However, it also means that it is easy to make changes to suit your particular needs. The P-code interpreter has an optional TRACE mode, which can be helpful both in debugging programs and in learning how things work. The manual covers the operation of the three programs, the differences from standard Pascal, and a few examples. It does not cover standard Pascal or provide any tutorial material.

Tiny Pascal is available for 3.0 and 4.0 PETs and the 8032 on 2040/4040 diskette for $35.00, and on cassette for $40.00. Tiny Pascal Plus+ is $50.00 on diskette and $55.00 on cassette. Abacus Software, P.O. Box 7211, Grand Rapids, MI 49510.

KMMM Pascal — AB Computers

KMMM Pascal was written by Willi Kusche of Wilserv Industries and is marketed by AB Computers. This implementation is probably closest to Tiny Pascal, but Boolean, real, and text variables are supported. The editor combines the screen-editing features of the PET with the features of a traditional line editor. The lack of line numbers makes it a little difficult to use the line commands. The compiler is in machine code and generates P-code, which is converted to executable machine code by a program called the "Translator." After this machine code is in place, a BASIC SYS command will run the compiled program.

The manual, supplied to me in preliminary form, includes a description of the operation of the editor, the differences from standard Pascal with a few examples, and a full set of syntax diagrams. Standard Pascal is not documented, nor is there any tutorial material.

KMMM Pascal is available for 3.0 or 4.0 PETs (including 8032) on 4040 or 8050 diskette for $85.00. An older version, which runs in 16K but doesn't support floating point numbers, is available for $75.00. AB Computers, 252 Bethlehem Pike, Colmar, PA 18915.

Commodore TCL Pascal

This version, written by Keith Frewin of Transam Components Limited, and marketed in the U.S. by Commodore as its "Pascal Development System," is a standard Pascal. The full power of the language is realized. Producing a source file is very much like writing a BASIC program, and auto line numbering, renumbering, searching, and changing capabilities have been added.

The compiler can be used in two ways. In the resident mode, the source, object, and compiler program are held in memory together. In the disk mode, the compiler, source, and object are all in files stored on disk.

The manual includes complete documentation of the editor and compiler. There's a lot of tutorial material, illustrating standard Pascal usage. However, I would not recommend that anyone learn Pascal from this manual.

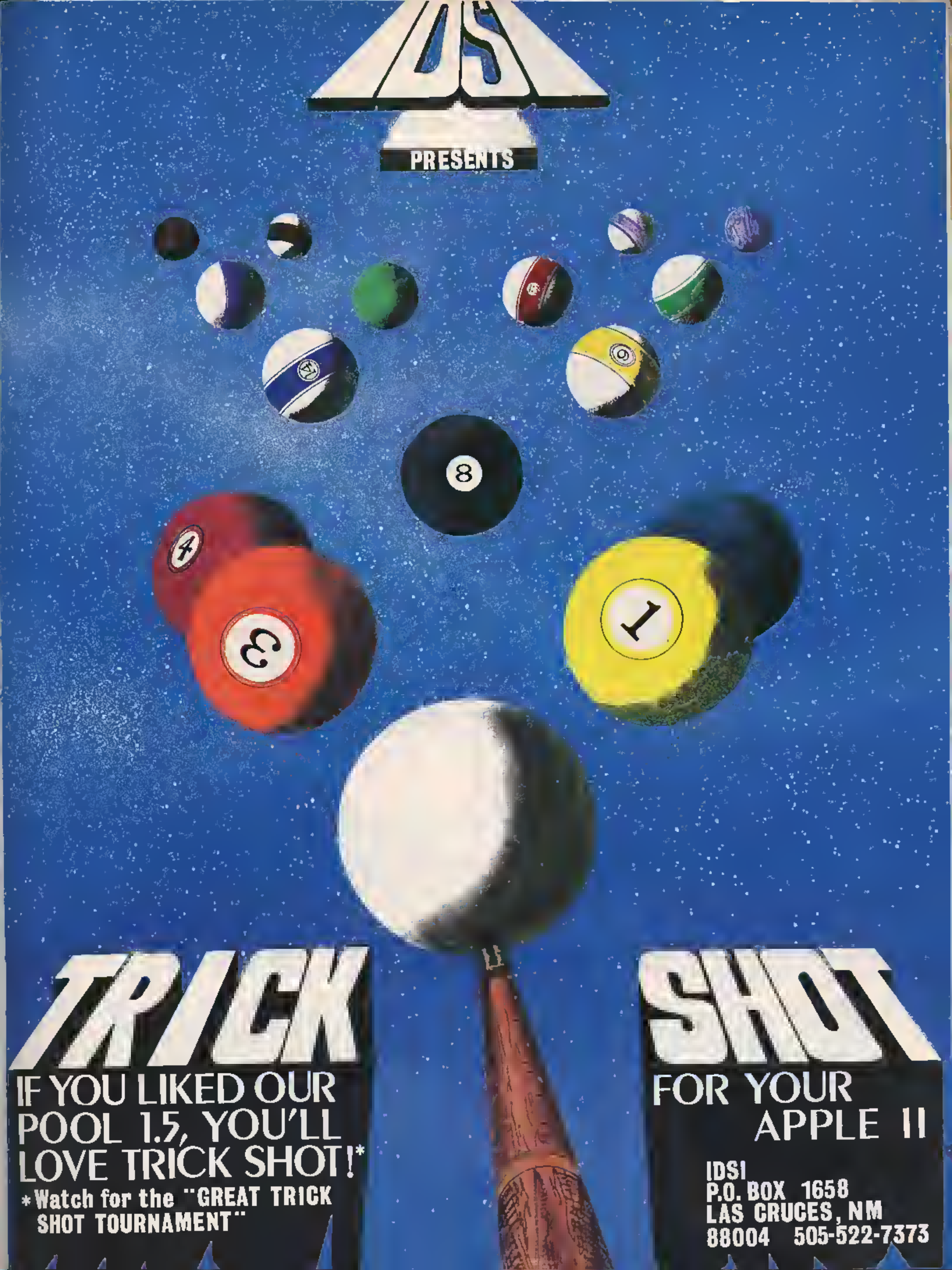
TCL Pascal is available for 8032 and 8050 with protection ROM and manual for $295.00. An older version may still be available for 3.0 ROM PETs and 2040 for $250.00.

Final Thoughts

There are three very different versions of Pascal available for the PET. They differ in price, "completeness," and in the machine configurations they require. Certainly the best is the TCL version, but it is definitely not over three times better than KMMM Pascal, as the price might indicate. Tiny Pascal is not as good as the other two, but it is the cheapest, it is modifiable, it may be the only version that works on your system, and it will fill the need for a teaching system.

According to sources, Commodore plans to offer a UCSD version of Pascal for the 8096 (not the SuperPET) sometime this year. Further details were not available.

MICRO


IDSI
PRESENTS
TRICK SHOT
IF YOU LIKED OUR POOL 1.5, YOU'LL LOVE TRICK SHOT!*
*Watch for the "GREAT TRICK SHOT TOURNAMENT"
FOR YOUR APPLE II
IDSI
P.O. BOX 1658
LAS CRUCES, NM
88004 505-522-7373

Submitting an article to MICRO?

The following tips will help both you and our staff:

Text

- Send typewritten, double-spaced copy.
- Put your name and address on the first page of the article, and name on each page.
- Provide a summary of the article, and a brief biographical note.
- Use tables and figures — they're not only effective, but add visual interest to your article.

Listings

Make sure *printed* listings are:

- machine generated
- black ink on white paper
- commented
- thoroughly tested

We'd appreciate listings on *magnetic media*. Please provide:

- BASIC and binary files
- assembler source files
- specifications and loading instructions

Write for a copy of our Writer's Guideline.

MICRO

From Here to Atari

Jim Capparell
297 Missouri
San Francisco, California 94107

This month I'm going to talk about the vertical blank (Vblank) interrupt. I will give you a working definition of what an interrupt is, then discuss how Vblank fits into the overall interrupt structure, what is accomplished in this time period, and how programmers may access this interrupt for their own use. I will also provide a simple program to illustrate the use of Vblank vectors and how to insert code at VVBLKD.

Recall last month, in my discussion of raster scan graphics, that the term vertical blank is given to that time period when the electron beam is turned off and returned to the upper left corner of the video screen, ready to start tracing a new frame. The number of machine cycles available at Vblank is some fraction of 29868 machine cycles that are needed to trace one entire television frame. In the normal graphics 0 (text screen), approximately 7980 machine cycles are left over at Vblank to be shared by the Operating System Vblank interrupt service routine (ISR) and any programmer supplied code. The term interrupt applies to any signal, originating from hardware or software, which serves to suspend normal mainline program flow.

When an interrupting event occurs, the program counter (PC) and processor status registers are automatically saved on the system stack. The processor then executes special code referred to as an interrupt service routine (ISR). The address of the ISR is found in a memory location reserved for this purpose, called an interrupt vector. When the ISR is finished, the values of the PC and status registers are retrieved from the stack and processing of the suspended program is resumed as if nothing had intervened. This all happens at machine speed — in hundreds of microseconds.

The vertical blank interrupt is an essential part of the Atari operating system and appears as a non-maskable interrupt (NMI) to the system. The NMI is only one of three possible interrupts that the Atari can process. These three, chip reset, NMI, and IRQ, are analyzed further by interrupt service software. Whenever an NMI or an IRQ signal occurs, the appropriate service routine is executed. These service routines interrogate a status register to isolate the interrupting source. See table 1 for a breakdown of vectors and contents for each type of interrupt.

It's apparent from table 1 that all NMI interrupts are vectored through location \$FFFA to the NMI interrupt service routine (ISR) starting at address \$E7B4. Since there are three possible causes of an NMI, the ISR must determine the source of the interrupt by interrogating an NMI status register at address \$D40F. This location, called NMIST in the documentation, has bit 7 set when a DLI occurs, bit 6 set when a Vblank occurs, and bit 5 set when the system reset button has been pressed. If neither a DLI nor a system reset caused the NMI, then a Vblank interrupt is assumed by the ISR and the processor jumps to the address contained in the vector at \$0222.

There are actually two vectors used by Vblank through which a programmer may introduce additional or replacement code. One vector, referred to as vertical blank immediate vector VVBLKI, is at address \$0222. This vector normally contains the address \$E7D1, the start of the system Stage 1 Vblank ISR. Should it be necessary to either replace system functions or simply perform operations prior to the system code, then you would use this vector. The other vector location, called vertical blank deferred VVBLKD, is at address \$0224. This vector normally contains the address \$E93E, which is the start of code for the system return from interrupt. The contents of \$0224 would be changed to point to new code when your operation was needed after system housekeeping was accomplished.

The Vblank process is actually divided into two stages. Whenever a Vblank NMI occurs, the following events always happen:

1. Processor registers A, X, and Y are pushed on stack.

2. Interrupt request is cleared by writing zero to \$D40F.

Table 1

```
INTERRUPT     VECTOR    ISR LOCATION

CHIP RESET    FFFC      E477
NMI           FFFA      E7B4
   Display list   Jump through   0200
   Vertical Blank                0222 and 0224
   S/Reset key                   E474
IRQ           FFFE      E6F3
   Serial bus output ready jump through        020C
   serial bus output complete                  020A
   Serial bus input ready                      020E
  *Serial bus proceed line                     0202
  *Serial bus interrupt line                   0204
  *Pokey timer 1                               0210
  *Pokey timer 2                               0212
  *Pokey timer 4(Bug in O.S. timer 4)          0214
   Keyboard key scan                           0208
   Break key                                   ????
  *6502 break instruction                      0206

* These vectors are unused by the O.S. and are initialized to
point to an RTI instruction.
```

```
                ; **  PROGRAM EXAMPLE 1 **
                ;PROGRAM SETS UP A VVBLKD ISR
                ;
                ; SET UP NEW VECTOR WITH A BASIC USR CALL A=USR(1536)
                ; NEED TO DO THIS WHENEVER SYSTEM IS RESET.

0000                      *=    $600       PUT IN PAGE 6 DECIMAL 1536
0600 68                   PLA              NULL VALUE FROM BASIC
0601 A907                 LDA   #7         INDICATOR FOR VVBLKD
0603 A206                 LDX   #06        HIGH BYTE FOR VECTOR ADDR
0605 A040                 LDY   #$40       LOW BYTE FOR VECTOR ADDR
0607 205CE4               JSR   $E45C         SET UP DEFERRED VECTOR
060A 60                   RTS              RETURN TO BASIC
                ; **   **   **   **
                ; ROUTINE AT DECIMAL 1600 IS DESIGNED TO WASTE TIME.
                ; PUT A NUMBER FROM 1 - 5 IN DECIMAL 1568.
                ; USE POKE 1568,N
                ; THIS IS THE ISR WHICH SIMPLY WASTES TIME.
0608                      *=    $640
0640 A600                 LDX   0           INIT COUNTERS
0642 A400                 LDY   0
0644 E8         LOOP1     INX              INCR COUNT
0645 EC2006               CPX   $620        DELAY VALUE
0648 F003                 BEQ   LOOP2
064A 18                   CLC               FORCE BRANCH
064B 90F7                 BCC   LOOP1
064D C8         LOOP2     INY
064E CC2006               CPY   $620        DELAY VALUE
0651 F003                 BEQ   EXIT        DONE ?
0653 18                   CLC               NO-FORCE BRANCH
0654 90EE                 BCC   LOOP1
0656 4C3EE9     EXIT      JMP   $E93E      TAKE NORMAL VBLANK EXIT
```


3. Jump through VVBLKI normally pointing to Stage 1 Vblank.

When Stage 1 processing is executed, it increments the three-byte counter called RTCLOK at addresses $12, $13, and $14. Location $12 is the most significant byte. This counter wraps to zero after approximately 77 hours and then continues counting. The attract mode is also processed at Stage 1; that is the process which causes the colors on your screen to start shifting if no key has been pressed on the keyboard in the previous nine minutes.

Additionally, system timer 1 at locations $218 and $219 is decremented if it is non-zero. When the counter goes to zero, an indirect JSR is performed *via* a vector at addresses $226 and $227. Note that an indirect JSR is performed by copying the address from the vector to the stack and executing an RTS instruction.

At this point a test is made to determine if a time-critical section of code was interrupted. If either the I bit in the processor status register or the critical flag at address $42 are set, then the interrupted code is assumed to be time-critical. When this occurs, the registers are restored and an RTI instruction is executed.

The critical flag can be set by a Serial I/O in progress. If no time constraints are present, then Stage 2 processing is begun. It is in this section of code that IRQ interrupts are enabled, keyboard auto repeat logic is processed, keyboard debounce is performed, and system timers 2, 3, 4, and 5 are processed. In addition, the color data for playfield and player/missiles are updated. This color data and other RAM locations, called shadow registers, are copied into their associated hardware locations. Stage 2 also reads the game controller data from Jacks 1, 2, 3, and 4 into RAM memory.

To insert code either at VVBLKI or VVBLKD, the address where the new code resides must be placed into the appropriate vector. A system routine is provided at $E45C. This routine insures that both bytes of the vector will be updated while Vblank is enabled. A vertical blank can be processed during a call to this routine. The routine is called SETVBV in the documentation and the calling sequence is:

reg A (update indicator)
= 1-5 then update timers 1-5
= 6 for immediate Vblank vector VVBLKI
= 7 for deferred Vblank vector VVBLKD

reg X = Most Significant Byte of vector address

reg Y = Least Significant Byte of vector address

JSR SETVBV Jump to subroutine

The A, X, and Y registers may be altered.

The display list interrupt will always be enabled on return.

A knowledge of processing interrupts and inserting code at interrupt vectors is essential to get the most from the Atari. With this example you should have enough information to experiment with the Vblank vectors. Interrupt-driven sound control, page flipping, animation techniques, greater color control, and many other procedures are possible.

MICRO

List Scroller

This program lets you scroll forwards or backwards through a listing to viaw any part of a BASIC program without requiring a series of keyed LIST commands.

Colin Macauley
39 Shoalhaven Street
Werribee
Victoria 3030
Australia

The problem with editing a BASIC program is that a LIST only displays a small "window" of your program. Often you wish to view outside this window, which requires a further LIST. This can be very frustrating because the screen depth never appears to be long enough to display all the program lines desired, especially on my OSI Superboard. Each subsequent LIST scrolls a lot, if not all, of your previous listing off the screen.

Word processors have the ability to scroll through a document, either forwards or backwards, to allow for operator insertions, deletions, etc. My program is similar in operation and allows continuous controlled forward or backward scrolling through a BASIC program *after* a LIST.

The keys used are "CTRL-A" for backward scrolling and "CTRL-Z" for forward scrolling. Holding down the keys will allow continuous scrolling until the keys are released. The program is located in RAM unused by the OSI BASIC interpreter, and is not lost by warm and cold starts to the system.

To use the program, the BASIC input vector ($218,$219) must be re-set by typing in the following:

POKE 536,34 :POKE 537,2<CR>

to divert keyboard input through my program. Thus, when you want to review your own BASIC program, a LIST (e.g. LIST 10) should be entered and the program listing on the screen manipulated, using the CTRL-A and CTRL-Z keys.

Operation of the program is difficult to follow unless you are fully conversant with the storage and tokenizing of BASIC program lines, and the routines involved in actually running a BASIC program. Essentially, the program directly manipulates the input buffer to cause a serial supply of single line LIST instructions to be entered and run while the previously described keys are depressed. Further information regarding BASIC operation, with references to specific subroutines in the 8K BASIC ROMs, may be found in *The First Book of OSI* by Jim Williams and George Dorner (Aardvark Technical Services), and *OSI BASIC in ROM* by Edward H. Carlson.

The sample run shows how the program works by scrolling forward through a BASIC program until line 150 (CTRL-Z) and backwards from line 150 until line 90 (CTRL-A).

Colin Macauley is a qualified physicist and a member of the firm of Callinan and Associates, Patent Attorneys. He uses a modified OSI Superboard II and is mainly interested in utility-type programming. His current interest is development of software for his brother's minimum chip 6502-based computer which has software-controlled video.

***Figure 1*: Sample Run**

```
100 FOR R=2 TO 0 STEP -1

110 FOR M=P1 TO P2:POKE M,Q:NEXT M

120 FOR M=P1+R TO P2 STEP 3:POKE M,T:NEXT M

130 V=R+1

140 FOR M=P1 TO P2

150 V=V-1

140 FOR M=P1 TO P2

130 V=R+1

120 FOR M=P1+R TO P2 STEP 3:POKE M,T:NEXT M

110 FOR M=P1 TO P2:POKE M,Q:NEXT M

100 FOR R=2 TO 0 STEP -1

90 FOR H=1 TO 2
```

Listing 1

```
0222             *=$0222
0222             ;ENABLE NSG PRINTER
0222 A94C        LDA #$4C
0224 8503        STA $03
0226             ;GET CHAR. FROM KEYBD.
0226 20BAFF      JSR $FFBA
0229             ;CHECK KEY
0229 C901        CMP #$01
022B F02D        BEQ UP
022D C91A        CMP #$1A
022F F001        BEQ DOWN
0231 60          RTS
0232             ;
0232             ;SCROLL THRU ASCENDING LINE NOS.
0232 8A          DOWN TXA
0233 48          PHA
0234 98          TYA
0235 48          PHA
0236             ;CHECK NEXT PROGRAM LINE
0236 A000        LDY #$00
0238 B1AA        LDA ($AA),Y
023A             ;END OF PROGRAM LINES?
023A C900        CMP #$00
023C D00E        BNE DOWN1
023E             ;NO,BRA TO GET UP SCROLL
023E C8          INY
023F B1AA        LDA ($AA),Y
0241 C900        CMP #$00
0243 D007        BNE DOWN1
0245             ;YES,RET. TO BASIC
0245 68          DOWN4 PLA
0246 A8          TAY
0247 68          PLA
0248 AA          TAX
0249 A900        LDA #$00
024B 60          RTS
024C             ;GET NEXT LINE NO.
024C A000        DOWN1 LDY #$00
```

```
024E C8          INY
024F C8          INY
0250 C8          INY
0251 B1AA        LDA ($AA),Y
0253 48          PHA
0254 88          DEY
0255 B1AA        LDA ($AA),Y
0257             ;LOAD INPUT BUFFER
0257 4CA602      JMP SCRO
025A             ;
025A             ;SCROLL THRU DESCENDING LINE NOS.
025A 8A          UP TXA
025B 48          PHA
025C 98          TYA
025D 48          PHA
025E             ;PUT PRESENT LINE NO. POINTER ADDR. IN $AA,AB
025E 2032A4      JSR $A432
0261             ;SAVE $AA,AB IN $DC,DD
0261 A5AA        LDA $AA
0263 85DC        STA $DC
0265 A5AB        LDA $AB
0267 85DD        STA $DD
0269 A000        UP3 LDY #$00
026B B1DC        LDA ($DC),Y
026D             ;END OF BASIC PROGRAM LINE?
026D C900        CMP #$00
026F             ;NO,CHECK IF OUT OF RANGE
026F D019        BNE UP1
0271 C8          INY
0272 B1DC        LDA ($DC),Y
0274             ;START OF PREVIOUS LINE?
0274 C5AA        CMP $AA
0276             ;NO,CHECK IF OUT OF RANGE
0276 D012        BNE UP1
0278 C8          INY
0279 B1DC        LDA ($DC),Y
027B C5AB        CMP $AB
027D D00B        BNE UP1
027F             ;SET UP LINE NO.
```

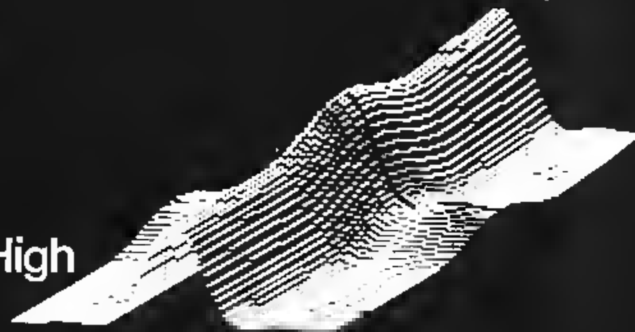
(Continued)

Listing 1 *(Continued)*

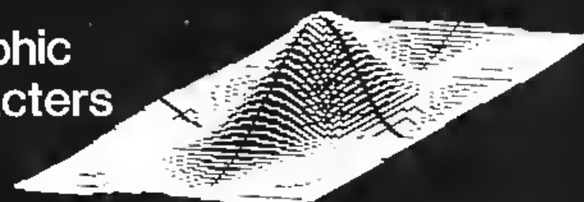
```
Ø27F C8       INY
Ø28Ø C8       INY
Ø281 B1DC     LDA ($DC),Y
Ø283 48       PHA
Ø284 88       DEY
Ø285 B1DC     LDA ($DC),Y
Ø287          ;LOAD INPUT BUFFER
Ø287 4CA6Ø2   JMP SCRO
Ø28A          ;TEST FOR OUT OF RANGE IE BELOW $Ø2FF
Ø28A A5DD     UP1 LDA $DD
Ø28C C9Ø2     CMP #$Ø2
Ø28E DØØ9     BNE UP2
Ø29Ø A5DC     LDA $DC
Ø292 C9FF     CMP #$FF
Ø294 DØØ3     BNE UP2
Ø296          ;YES,RET. TO BASIC
Ø296 4C45Ø2   JMP DOWN4
Ø299          ;BUMP SEARCH ADDR.
Ø299 C6DC     UP2 DEC $DC
Ø29B A5DC     LDA $DC
Ø29D C9FF     CMP #$FF
Ø29F DØC8     BNE UP3
Ø2A1 C6DD     DEC $DD
Ø2A3          ;LOOP TO FIND END OF LINE
Ø2A3 18       CLC
Ø2A4 9ØC3     BCC UP3
Ø2A6          ;
Ø2A6          ;LOAD INPUT BUFFER
Ø2A6          ;LINE NO. LO IN Y, HI IN A
Ø2A6 A8       SCRO TAY
Ø2A7 68       PLA
Ø2A8          ;CONVERT TO FLOATING POINT
Ø2A8 2ØC1AF   JSR $AFC1
Ø2AB          ;CONVERT TO ASCII STRING
Ø2AB 2Ø6EB9   JSR $B96E
Ø2AE          ;SET UP POINTERS TO $BC ROUTINE
Ø2AE A912     LDA #$12
Ø2BØ 85C3     STA $C3
Ø2B2 A9ØØ     LDA #$ØØ
Ø2B4 85C4     STA $C4
Ø2B6          ;LIST TOKEN
Ø2B6 A999     LDA #$99
Ø2B8 8513     STA $13
Ø2BA A2ØØ     LDX #$ØØ
Ø2BC          ;ASCII STRING TO BUFFER
Ø2BC BDØ1Ø1   DOWN3 LDA $Ø1Ø1,X
Ø2BF C9ØØ     CMP #$ØØ
Ø2C1 FØØ6     BEQ DOWN2
Ø2C3 9514     STA $14,X
Ø2C5 E8       INX
Ø2C6 18       CLC
Ø2C7 9ØF3     BCC DOWN3
Ø2C9          ;INSERT NULL
Ø2C9 9514     DOWN2 STA $14,X
Ø2CB          ;DISABLE MSG. PRINTER
Ø2CB A96Ø     LDA #$6Ø
Ø2CD 85Ø3     STA $Ø3
Ø2CF 68       PLA
Ø2DØ A8       TAY
Ø2D1 68       PLA
Ø2D2 AA       TAX
Ø2D3 86DC     STX $DC
Ø2D5          ;RESTORE SP TO AVOID STACK OVERFLOW
Ø2D5 A2FC     LDX #$FC
Ø2D7 9A       TXS
Ø2D8 A5DC     LDA $DC
Ø2DA AA       TAX
Ø2DB          ;BASIC EXECUTION ROUTINE
Ø2DB 4CF6A5   JMP $A5F6
```

MICRO

GRAPHICS
FOR OSI COMPUTERS



☆ You Can Produce The Images Shown Or Yours And Program Motion With Our 256 By 256 High Resolution Graphics Kit. Thats 65,536 Individually Controlled Points On Your TV Screen.



☆ Increase Column/Line Display. You Can Set Up Your Own Graphic Pixels Including Keyboard Characters And Unlimited Figures.

☆ This Kit Includes All Parts, Software And Assembly Instructions Required To Get Up And Running. The Included 8k Of 2114 Memory Is Automatically Available When Not Using The Graphics. Boot Up And See 8k More Memory.

☆ Adding The Kit Does Not Affect Your Existing OSI Graphics. Use Both At The Same Time Or Separately.

☆ Buy The Entire Kit, Including Memory, For $185.00 Or A Partial Kit For Less If You Have Parts. Board And Instructions $40.00. Instructions Include Software.

For This Kit Or A Catalog Of Other Kits, Software And Manuals Call Or Write:

MITTENDORF ENGINEERING
905 Villa Neuva Dr.
Litchfield Park, Az. 85340
(602)-935-9734

MICRO

The Single Life

Editor's note: This column is a new, on-going addition to MICRO. The purpose is to provide information about, product evaluations of, and uses for the single board computers and their makers. Readers are encouraged to submit entries for publication to:

Brad Rinehart
1508 Stanton St.
York, Pennsylvania 17404

So, you've put away your old single board computer and have purchased one of the new personal computers. As you unpack it from the carton you marvel at the simplicity and ease with which you can plug it in, turn it on, and find something displayed on the screen. Not only that, but it comes complete with pre-packaged software! Certainly more appealing than the pile of bread-boards and wire lying in the corner. Then suddenly you begin to reminisce about the "good old days" when adding a new peripheral to your single board machine left a feeling of pride and accomplishment. Well, you're not alone.

Those days are behind us now. In today's world, plastic packaging, pre-packaged software, and intriguing games have taken the spotlight. What most of us have failed to realize though, is that most of the single board promoters are still with us and are producing some pretty sophisticated hardware and software. Some of these manufacturers even custom tailor their hardware and software for OEM and industrial applications.

Many of these single board-based machines have found their way into schools, laboratories, institutions, industry, and of course in the homes of computer hobbyists. OEMs may well want to consider the single board computer for its versatility, adaptability, durability, and modular design. Many manufacturers produce:

Backplanes
Memory boards
Disk controllers
EPROM cards
RS-232, IEEE-488, and parallel interfaces
A/D and D/A converters
Cabinets and enclosures

and a host of other items all designed to expand and support one or more of the current single board computers. This type of modularity allows OEMs and industrial manufacturers to include necessary design functions in their products without incurring the expense of unwanted or unnecessary hardware.

For example, Hudson Digital Electronics Inc. (HDE Inc.) supports their line of TIM-, AIM-, SYM-, and KIM-compatible (hence the name TASK Masters) hardware with complete development systems.

In comparison with the personal computer manufacturers, the single board people rely heavily upon OEMs and industrial users to purchase their products. Therefore, their products are usually built to be durable, reliable, sophisticated, and easy to maintain. This explains the higher initial cost, as compared to the personal computer, which is normally built entirely on a single board and packaged to please the consumer. However, when one considers the advantages to the industrial user of modular design, this higher initial cost factor can easily be justified. For example, some of the single board systems can be expanded from a single five-inch disk drive to several double-sided eight-inch drives simply by changing the disk controller card and drives. The software remains compatible. In addition, because most do *not* use memory-mapped video, almost any CRT or hardcopy terminal can be interfaced directly to them. This does create a problem of providing software which will interface to different brands of terminals, but this problem can be overcome, as we shall see later.

The problem of maintenance is greatly simplified in a modular system. To isolate a problem, one must only remove and replace the individual boards until the problem is corrected. Then, the board that caused the problem is either sent for repair or repaired by the user. In many cases a modular system will still be operable even if no spares are available to replace the broken board. For example, if the problem were in a memory board, the system might still operate, using the memory available on other memory boards which had not failed. Hence, very little down time!

If the user does stock spare boards, the cost is usually very minimal. Then, after the problem has been isolated, the broken board may be returned to the manufacturer for repair. In addition, Perry Peripherals on Long Island does repair work on AIM, SYM, and KIM boards. (Steve Perry told me that the turn-around time is approximately one week.)

Because the single board promoters depend on those who use their products in custom applications, they are most eager to provide excellent customer support. HDE, for example, provides in-house engineering and custom software development on a contract basis. Other manufacturers such as The Computerist, Micro Technology Unlimited, Systems Innovations Inc., etc., provide detailed explanations describing how to interface and use their products. Also helpful are Perry Peripherals' application notes concerning the AIM, KIM, and SYM single board machines.

The single board people seem to have taken a silent oath to standardize the bus structure of their products as much as possible. Therefore, it may be possible to use one manufacturer's processor board with another's card cage and memory cards. In yet another step, some vendors supply interfaces which will adapt one manufacturer's product to another's. For example, Perry Peripherals can provide a KIMSI to HDE's disk system interface. This idea of modular design and interfacing one manufacturer's product to another erases the problem of obsolescence. So, when a new board is introduced, just plug it in. There is no need to replace the entire system in order to upgrade it.

In addition to the equipment manufacturers themselves, other vendors are providing hardware. Optimal Technology offers an EPROM programmer, complete with software listings, that interfaces through a PIA to many of the single board systems. Keystone Data Consultants Inc. will be releasing a time-of-day clock for the KIM 4 bus (2nd quarter 1982). In addition, Keystone Data can provide equipment and software to interface 110V and 220V devices to numerous single board machines. Progressive Computer Software Inc. offers a disk head cleaning kit for HDE disk systems. The list goes on and on.

Why have the personal computers attained so much popularity while the single boards have been "left in the dust?" Good question! There appear to be several reasons. As mentioned

before, initial purchase price has an effect upon the "business user." In today's runaway inflation, the business man is out to get the most for his hard-earned dollar. Another reason may be packaging. The personal computers are packaged in an appealing plastic and sometimes metal cabinet. Many single board systems are packaged in a heavy, industrial cabinet, enough to give any secretary heart failure. I have also been told that because very few "how-to-fix-bugs-in-your-system" articles have been written about the single boards, they have not received the publicity of some other systems. Perhaps some of the single board manufacturers should intentionally build boards that need fixing! The most obvious reason for this lack of interest though, is the apparent lack of software for these single board-based systems.

I find this hard to believe. After all, weren't the single boards here first? Don't those of us who suffered through the "hex keypad syndrome" pride ourselves in having to really "know the system" in order to have written software? Although I will admit it would have been very difficult to write "General Ledger" on a KIM with a hex keypad, times have changed! One of the oldest and probably best-known supporters of the TIM, AIM, SYM, and KIM single boards, HDE Inc., offers a great deal of development software to complement their tried and proven disk operating system — FODS.

For assembly language programmers, there is a text editor, assembler, dynamic debugging tool, and a disassembler that generates source code, complete with labels. The text editor may be used, not only to generate assembly language source code, but also for composition.

The assembler is a multifile assembler, meaning that source code is not limited to the amount of available memory. Several files may be assembled by calling them in from the disk. In addition, the assembler includes a formatter which prints the listings in the proper format, therefore alleviating this chore when the source code is entered by the programmer. This, of course, saves several bytes of memory per line of source code (no need to insert spaces to align the columns).

The dynamic debugging tool (DDT) is a comprehensive debugger that allows the assembly language programmer to insert break points in a program, single-step the program, proceed to a breakpoint, step multiple instructions at a time, and read and change memory locations. In addition, DDT will display memory as a single byte, an address word, ASCII characters, or disassembled instructions. Memory may be changed using the current display mode. For example, if an ASCII 'A' is displayed, to change it to a 'B' it is only necessary to type in a 'B'.

While in the instruction disassemble mode, the instruction may be changed by typing in the mnemonic and data for the new instruction. The user should be aware that if an instruction's byte count is longer than that of the instruction being replaced, no attempt is made to move the existing code. Remember, this is a debugger, not an assembler. In the single-step mode (this is a software function, not hardware) DDT displays the contents of all registers, the instruction address and mnemonic, as well as the condition code flags, after each instruction is executed.

The disassembler (AID for Advanced Interactive Disassembler) is truly a work of art. Most disassemblers merely generate hard copy listings that contain the instruction mnemonic and referenced address. If you've ever taken one of these listings and tried to decipher it, you'll really appreciate AID. Not only does it assign labels to all the referenced addresses and output hard copy, it also builds a source file in memory that is compatible with the HDE text editor (TED). In addition to all this, if the disassembled source file grows larger than the available memory, AID asks you if you want to save the file to disk. If so, it automatically saves it, resets the textfile pointers, and continues disassembling.

All HDE-supplied development software interfaces to FODS, the File Oriented Disk System. FODS is a complete disk operating system, because simple commands are provided for all disk operations. User commands may be added by building an object module (executable 6502 code) and saving the module to disk identified by a three-character name. Then, whenever the three characters are entered from the keyboard, FODS will access the system drive, load the module, and execute it. In addition, whenever FODS reads or writes the disk, it does a read after read, or read after write verify. In other words, it compares what was read from, or written to, disk with the data in memory, thus insuring positive data transfer.

Disk accesses made through FODS are very fast. This is partly due to the format used to save files to disk. FODS does not use the sector-mapping technique when writing to the disk. Instead, disk files are written one sector after another, in a straight line, so to speak. This can cause some grief to the user who is accustomed to sector-mapping techniques that write data to any available sectors. So, as new files are created and old ones deleted, "holes" appear in the index. When the disk becomes full it is necessary to "PAK" it, thus filling in the gaps. From a development standpoint, this can be somewhat annoying. In actual applications programs, however, this is of little consequence as methods have been devised to re-use the disk space without "PAK"ing the disk.

One good point is that by allowing these holes, deleted files can be recovered — provided the disk has not been packed. Have you ever deleted the wrong file and wasted hours of development time? Well, under FODS, you can recover these.

In addition to assembly language development tools, there are numerous high-level languages available for HDE's disk-based systems. Eric Rehnke's 6502 FORTH has been offered in cassette version for some time now. I understand that it either is or will soon be avilable for HDE disk-based systems.

HDE is offering CPM on their systems. Yes, a 6502 version of the popular CPM operating system. In addition, a UCSD Pascal interface is available from HDE or their distributors. This package will interface Softech Microsystem's popular languages, Pascal, FORTRAN, and BASIC, to HDE's disk system. An advantage to using the Pascal system is that your programs become portable. This means that a package written on another machine operating under the UCSD system will normally run under the HDE disk system, and *vice versa*. This really opens up some doors for software compatibility!

Probably the most significant addition to HDE's growing line of development software is HDE Disk BASIC. Built around the popular Microsoft BASIC, this package is really an eye opener. I have been evaluating HDE Disk BASIC for approximately a year now, and believe me, an enormous amount of work and forethought has gone into this package.

In order to allow for the multitude of terminal configurations that are on the market today, a personality module has been implemented which allows the user to custom tailor the software to the terminal. Also, a 250-character input buffer allows numerous statements per line. HDE BASIC allows the user to save data three ways. There is the simple "snapshot" data file, as well as sequential data files, and random access files. All of the old KIM BASIC commands have been carried along in this new version, making it an easy task to upgrade your old software. In addtion, a multitude of new commands have been added, making this one of the most powerful BASIC interpreters I have had the pleasure of evaluating.

OK, you say, this is all well and good, but I don't want to develop all my own software. Besides, I'm an industrial user and I need a multitasking process control system. Well, look no further! Keystone Data Consultants Inc. has such a system available. The software, called PECO, for Process Environment Operating System, provides the industrial user with a real time multitasking operating system. Keystone Data will configure the software to run stand-alone or in conjunction with an HDE disk-based system.

PECO allows job scheduling, queue processing, event timing, single or multiple access points, interrupt-driven operations, time-of-day scheduling and a host of other features. PECO is not a "pre-packaged" piece of software; Keystone Data provides the software based upon user specifications. The system can be provided in EPROM, or on disk. In addition, the software may be licensed in either object or source code.

There are numerous vendors willing to provide custom software for the industrial and OEM user. In many instances, equipment manufacturers themselves provide this service. Outside sources, like Keystone Data Consultants offer this service either to the manufacturer or their customers. Most single board manufacturers can provide the user with names of reliable software vendors.

What about applications software? Even though HDE Disk BASIC has only been released a short time, there are already several vendors supplying applications software and numerous user's groups providing games and numerous utility programs. The HDE Disk Program Library is presently undergoing some structural changes, but should be in operation by the time this is printed.

For the more serious user, I know of at least two vendors offering business applications programs for HDE disk-based systems. Western New York Micro offers Mail Manager, Mini Money Manager, Payroll Office, Memo Writer, and Tax Advisor. Keystone Data Consultants offers Data Foreman, Checking Account Management, Inventory Management, and will be releasing Accounts Payable/Receivable, and General Ledger. Contact both of these vendors for additional information.

The allure of plastic and chrome is very tempting. But, now that you know, give the single boards 'Another Look.' You may be surprised at what you find.

Manufacturers are encouraged to keep Mr. Rinehart aware of new products for the single board computers.

Names and Addresses of Vendors

Eric Rehnke
1067 Jadestone Lane
Corona CA 91720

Hudson Digital Electronics Inc.
P.O. Box O
Allamuchy, NJ 07820

Keystone Data Consultants Inc.
P.O. Box 606
York, PA 17405

Micro Technology Unlimited
2806 Hillsborough St.
P.O. Box 12106
Raleigh, NC 27605

Optimal Technology
Blue Wood 127
Earleysville, VA 22936

Perry Peripherals
P.O. Box 924
Miller Place, NY 11764

Systems Innovations Inc.
P.O. Box 2066
Lowell, MA 01851

The Computerist
34 Chelmsford St.
P.O. Box 6502
Chelmsford, MA 01824

Western New York Micro
P.O. Box 84
East Amherst, NY 14051

Some Help for KIM

Part 3

The author presents hardware and software for an improved single-step function. Also included is a TRACE routine.

Wayne D. Smith
Box 8352
Austin Peay State University
Clarksville, Tennessee 37040

Last month we examined the operation of the KIM single-step hardware. We saw that a simple modification to this hardware would allow the substitution of user-supplied software for the KIM software.

There is a number of variations on this basic scheme. Several K areas may be ANDed together to allow moving the single-step program to different areas as the need arises. Switches may be added to allow the selection of various K areas, as desired. If you extend this concept to the most general case, the circuit in figure 1 can be used. With these eight switches installed, the single-step software may be moved to anywhere in the lower 6K of memory.

By turning on the appropriate K switch, any K area can be prevented from generating the single-step NMI. Be sure that the K7 switch is also on, however, or the KIM software will single-step. If only the K7 switch is on and the NMI vector is set to $1C00, the system will perform as an unmodified KIM. If a K switch is on, it means that programs in this area will run at normal speed, even if the single-step switch is on. Since the K6 and K7 areas contain KIM ROM, these switches would both normally be on. They may be turned off, however, if you want to single-step any of the KIM software.

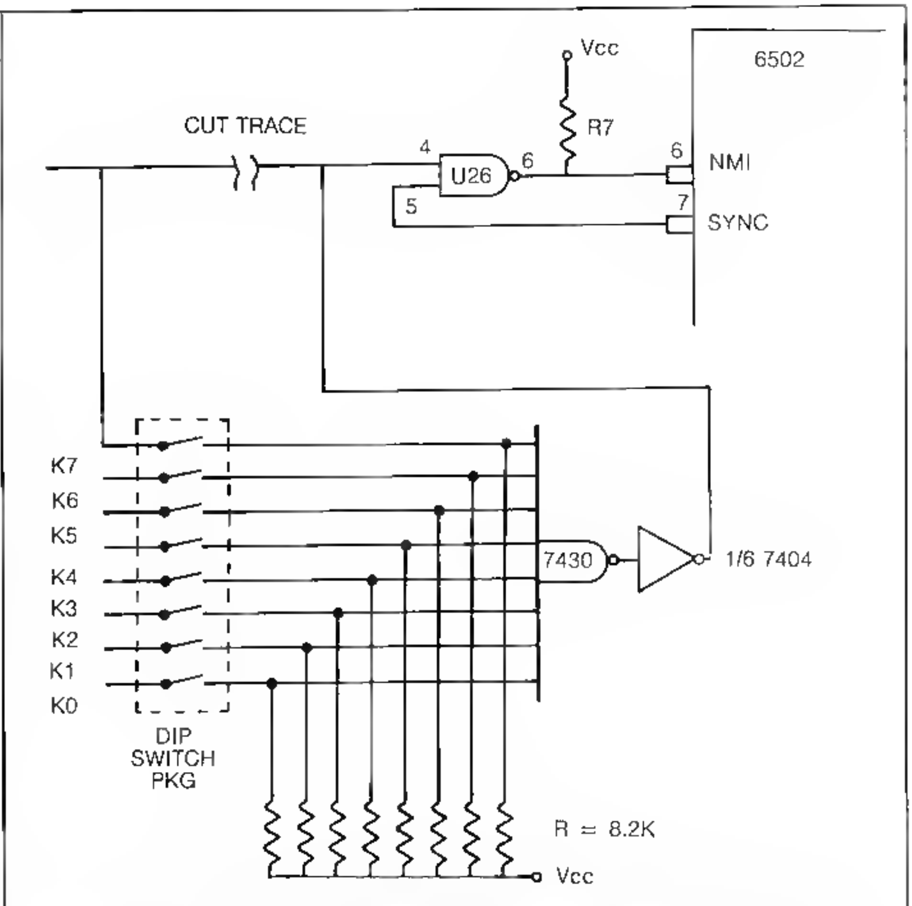


Figure 1: **The extension of the technique to include switch selectable K areas that will allow the use of a single-step program stored in any of several K areas.**

The 8.2 K Ω pull-up resistors shown in figure 1 are required to prevent noise, which might inadvertently generate a NMI if the 7430 inputs were left floating. These pull-ups supplement rather than replace the 560 Ω pull-ups already on the K signal outputs from the 74145. The additional pull-ups must be used, even though they do reduce the fanout of the K signals slightly. With this modification, try to limit the load on the K signal to about five normal TTL loads.

A secondary problem with the KIM software is eliminated with the hardware modification described above. This problem has to do with the KIM Peripheral Interface Adapters (PIAs). Included within the KIM single-step software are provisions for resetting the directional registers associated with the KIM PIA. It is, therefore, impossible to single-step a program which depends upon the setting of this directional register. As the single-step interrupt is generated, the KIM software

resets the registers to the KIM configuration. This, of course, negates any setting of the directional registers that may have been accomplished by the user. This can result in a program which runs correctly in the normal mode, but which will not run in the single-step mode.

Installing the switch-selected single-step area is probably best accomplished on a small perf board. I installed mine on the board that contains my data bus buffers. If you are willing to settle for the single additional area, the hardware modification can be made directly on the KIM board. A single trace must be cut as shown in figure 2. The AND gate (7408) may be mounted on the KIM board in any convenient location using a little contact cement. Mount the IC with the legs up. By being careful, you can make all the solder connections directly to the socket contacts on the KIM board. Just be sure that the solder doesn't interfere with the insertion of the KIM into the sockets. For convenience, the 7408 connections are listed in table 1.

The improved single-step program is shown in listing 1. The program is completely relocatable, and may be moved anywhere in memory without address changes. Other than the normal KIM page zero locations, SSTEP uses one additional page zero location, $EE.

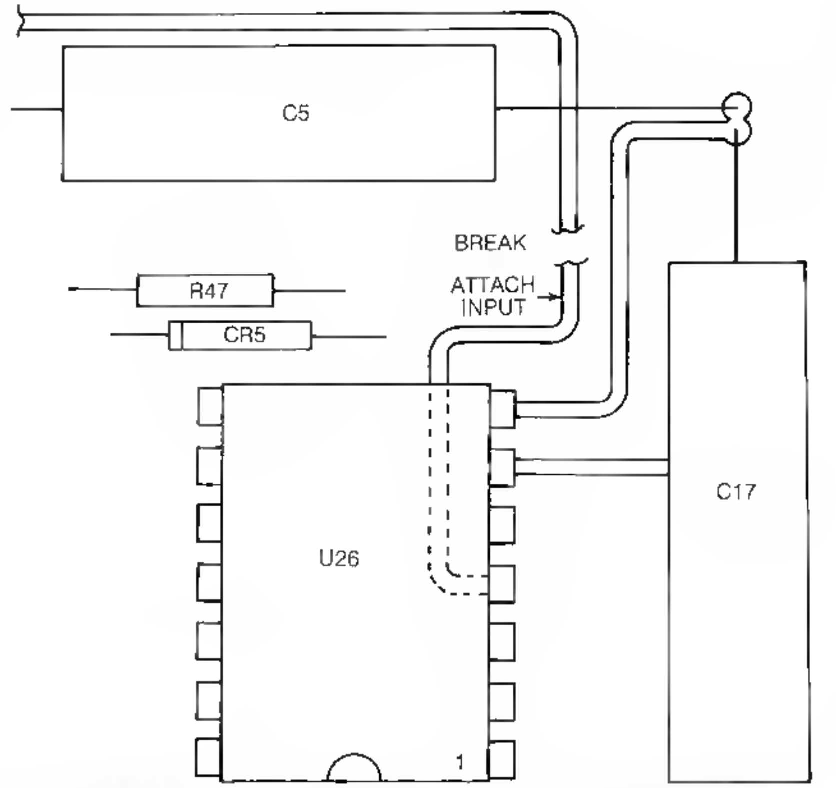


Figure 2: **Modification to the KIM-1 board for additional single-step programming areas. Only one trace must be broken as indicated in the drawing. The area shown is the lower right side of the board, near the keyboard. You may want to delay the modification until the 90-day warranty has expired.**

Table 1: **The 7408 pin connections for a single additional single-step program storage area. K5 is used as the additional area, but may be changed if desired. Pins not listed are not connected.**

Pin Number	Connection	Attach To
1	K5	KIM App J
2	K7	KIM App H
3	U26 pin 4	at cut trace
7	Ground	KIM Exp 21
17	Vcc	KIM Exp 22

If you use the program in the K5 RAM, resist the temptation to make any changes that lengthen the program. SSTEP ends at $17E6, and the KIM tape routines use variables at $17E7 and up. (Believe me, this wasn't easy, and accounts for some of the strange coding.) If you lengthen the program, the tape-write routine (including Super-tape) will destroy some bytes above $17E7, and render loading the program from tape impossible.

To use the program, first make the hardware modification as shown in figures 1 and 2. Then load the program normally. Be sure to set the NMI vector ($17FA, $17FB) to the starting location of the new single-step program ($1780). Now whenever the single-step switch is on, the complete status of the machine is printed as each instruction is executed. The status printed is the status *after* the instruction on that line has been executed. The registers are listed in the order X, Y, A, S (stack-pointer) and P (status).

The program shares one idiosyncrasy with the KIM single-step program. If a subroutine jump is executed to a routine within the non-interrupt area (say K7), the program will perform this routine at normal speed, without any output. The program will, however, also execute the first step following the return at normal speed. Recall that the interrupt is generated as the instruction is fetched, but not honored until the instruction has been completed. That means that the first instruction after the return generates an interrupt, but will not be listed. The status of the registers listed after the first step in the subroutine will actually be the status after the first step following the return is executed. The operand field should be ignored in this case. If this presents a problem, simply use a NOP immediately after any subroutine call to a non-interrupt area.

Listing 1: **Single-Step Program**

```
                        ;
                        ;  PROGRAM TO PROVIDE IMPROVED SINGLE STEP
                        ;  OPERATION OF THE KIM-1 MICROCOMPUTER.
                        ;  THE PROGRAM PROVIDES ALL THE NORMAL
                        ;  KIM-1 SINGLE STEP OPERATIONS PLUS
                        ;  PRINTING THE OPERAND FOR EACH INSTRUCTION
                        ;  AND PRINTING THE REGISTER CONTENTS
                        ;  OF ALL REGISTERS AFTER THE INSTRUCTION
                        ;  IS EXECUTED.  A HARDWARE MODIFICATION OF
                        ;  THE KIM BOARD IS REQUIRED IN ORDER TO USE
                        ;  THIS PROGRAM.
                        ;
                        ACC    =#$F3   STORAGE FOR ACCUMULATOR
                        PREG   =#$F1   STORAGE FOR STATUS REGISTER
                        PCL    =#$EF   STORAGE FOR PGM CNTR (LOB)
                        PCH    =#$F0   STORAGE FOR PGM CNTR (HOB)
                        YREG   =#$F4   STORAGE FOR Y REGISTER
                        XREG   =#$F5   STORAGE FOR X REGISTER
                        SPUSER =#$F2   STORAGE FOR STACK POINTER
                        POINTH =#$FB   ADDR OF CURRENT INST (HOB)
                        POINTL =#$FA   ADDR OF CURRENT INST (LOB)
                        TEMP   =#$EE   BYTES TO PRINT (NON KIM)
                        OUTSP  =#$1E9E KIM ROUTINE TO PRINT SPACE
                        PRTBYT =#$1E3B KIM ROUTINE TO PRINT A BYTE
                        SHOW   =#$1DAC REENTRY POINT FOR KIM LOOP
                        ;
1780                           *=#$1780  START OF ROUTINE (RELOCATABLE)
1780  D8                SSTEP  CLD            CLEAR DECIMAL MODE
1781  85 F3                    STA ACC        SAVE ACCUMULATOR
1783  68                       PLA            RECOVER STATUS AND
1784  85 F1                    STA PREG       SAVE IT.
1786  68                       PLA            RECOVER PROGRAM COUNTER,
1787  85 EF                    STA PCL        AND SAVE IT.
1789  68                       PLA            SAVE BOTH LOB AND
178A  85 F0                    STA PCH        HOB.
178C  84 F4                    STY YREG       SAVE Y REGISTER
178E  86 F5                    STX XREG       SAVE X REGISTER
1790  BA                       TSX            TRANSFER STACK POINTER
1791  86 F2                    STX SPUSER     TO X AND SAVE IT.
1793  20 9E 1E                 JSR OUTSP      PRINT A BLANK SPACE
1796  A2 04                    LDX #04        SET FOR 4 SPACES TO PRINT
1798  38                       SEC            FIND DIFFERENCE BETWEEN
1799  A5 F0                    LDA PCH        OLD AND NEW PROGRAM
179B  E5 FB                    SBC POINTH     COUNTER TO DETERMINE
179D  A5 EF                    LDA PCL        NUMBER OF BYTES
179F  E5 FA                    SBC POINTL     TO PRINT.
17A1  85 EE                    STA TEMP       SAVE THIS DIFFERENCE
17A3  A1 F6                    LDA (POINTL-4,X)  LOAD OP CODE
17A5  48                       PHA            AND SAVE IT.
17A6  29 0F                    AND #$0F       TEST FOR BRANCH AND OTHER
17A8  D0 0D                    BNE SETY       UNUSUAL DIFFERENCE INSTS.
17AA  68                       PLA            RECOVER OP CODE
17AB  C9 20                    CMP #$20       IF OP = 20 THEN
17AD  F0 08                    BEQ SETY       THREE BYTE INST.
17AF  29 90                    AND #$90       MASK ALL BUT B7, B4
17B1  F0 15                    BEQ SPACES     IF BOTH OFF, 1 BYTE INST
17B3  A9 02                    LDA #02        ELSE SET FOR 2 BYTE INST
17B5  85 EE                    STA TEMP       BY SETTING TEMP=2
17B7  A0 01             SETY   LDY #01        SET OFFSET TO OPERAND BYTE
17B9  C4 EE             AGN    CPY TEMP       COMPARE BYTES PRINTED TO
17BB  F0 0D                    BEQ SPACES     BYTES TO PRINT, IF NOT
17BD  B1 FA                    LDA (POINTL),Y  DONE, LOAD OPERAND
17BF  20 3B 1E                 JSR PRTBYT       THEN PRINT IT.
17C2  A0 02                    LDY #02        SET Y FOR SECOND BYTE
17C4  CA                       DEX            DECREMENT SPACES TO
17C5  CA                       DEX            PRINT BY 2.
17C6  D0 F1                    BNE AGN        IF NOT 0, THEN REPEAT
17C8  E8                SPACES INX              ONE MORE SPACE TO PRINT
17C9  20 9E 1E          MORESP JSR OUTSP      PRINT SPACES TO FILL
```

A Trace Program

While the single-step routine is a great help in debugging programs, there are times when it is inconvenient to sit at the terminal, pressing the G key after every step. It is often desirable to let the program run from beginning to end without intervention, but still have the steps traced, and the register information printed as the program executes. The resulting printout can then be analyzed at a more convenient time or place.

```
17CC  CA                 DEX              OPERAND FIELD.
17CD  D0 FA              BNE MORESP
17CF  A2 05              LDX #05          SET X TO PRINT 5 REGISTERS
17D1  B5 F0       AGAIN  LDA PREG-1,X       LOAD AND PRINT REGISTER
17D3  20 3B 1E           JSR PRTBYT       STORAGE IN THE ORDER:
17D6  20 9E 1E           JSR OUTSP          X, Y, A, STACK POINTER,
17D9  CA                 DEX                AND STATUS WITH SPACES.
17DA  D0 F5              BNE AGAIN
17DC  A5 EF              LDA PCL          UPDATE POINTH AND POINTL
17DE  85 FA              STA POINTL         FOR NEXT INSTRUCTION.
17E0  A5 F0              LDA PCH
17E2  85 FB              STA POINTH
17E4  4C AC 1D           JMP SHOW         AND RETURN TO KIM
17E7                     .END
```

Listing 2: Trace Addtion to Single-Step Program

```
                  ;
                  ; A NEW END TO CONVERT THE IMPROVED KIM-1
                  ; SINGLE STEP PROGRAM TO A TRACE PROGRAM.
                  ; THE PROGRAM OPERATES SIMILAR TO THE
                  ; SINGLE STEP PROGRAM EXCEPT THAT THE
                  ; PROGRAM BEING EXECUTED RUNS WITHOUT
                  ; OPERATOR INTERVENTION. A JUMP TO THE
                  ; KIM MONITOR WILL TERMINATE THE TRACE.
                  ; NOTE THAT THE ADDRESSES ARE RELATIVE TO
                  ; THE SINGLE STEP PROGRAM. THE PROGRAM
                  ; CAN NOT BE LOADED AT THE LOCATION SHOWN.
                  ; SEE TEXT FOR DETAILS.
                  ;
                  POINTL =#$FA   ADDRESS OF NEXT PROGRAM STEP
                  GOEXEC =#$1DC8 KIM EXECUTE ROUTINE ADDRESS
                  PRTPNT =#$1E1E KIM PRINT ROUTINE
                  CRLF   =#$1E2F KIM CARRAGE RETURN ROUTINE
                  OUTSP  =#$1E9E KIM ROUTINE TO PRINT A SPACE
                  ;
17E4                     *=#$17E4
17E4  20 2F 1E    START  JSR CRLF         START A NEW LINE.
17E7  20 1E 1E           JSR PRTPNT       PRINT ADDR OF NEXT INST.
17EA  20 9E 1E           JSR OUTSP        PRINT BLANK SPACE.
17ED  A0 00              LDY #00          SET INDEX FOR OP CODE
17EF  B1 FA              LDA (POINTL),Y   LOAD OP CODE AND
17F1  20 3B 1E           JSR PRTBYT       PRINT IT.
17F4  20 9E 1E           JSR OUTSP        PRINT TWO BLANKS.
17F7  20 9E 1E           JSR OUTSP
17FA  4C C8 1D           JMP GOEXEC       EXECUTE NEXT INSTRUCTION.
17FD                     .END
```

The program shown in listing 1 can be easily modified to perform this TRACE function, provided that it is stored in some location other than K5. As mentioned above, attempting to lengthen the program in K5 will result in erroneous operation of the tape routines, the single-step program, or both.

Since the initial portion of the TRACE program is identical to the SSTEP, only the step at $17E4 must be changed, and eight new steps added (listing 2). If desired, a flag can be set, and a test and branch sequence inserted at the location of the JMP SHOW instruction. In this manner, one program can accomplish both functions. I elected not to do this, since I keep SSTEP in K5 at all times, and only load TRACE when it is needed.

To insure that the program being traced terminates properly, simply include a JMP KIM (4C 64 1C) as the last step in the program. Needless to say, the trace program is of limited usefulness unless a hard copy terminal is being used. If you are using a CRT, you will just have to punch G and watch Star Trek at the same time (multiprocessing?).

MICRO

KEYSORT for BASIC 4.0

Gordon Campbell, Willowdale, Ontario, Canada

One of the most powerful utility programs published for the PET appeared in MICRO (23:11) — Rev. James Strasma's KEYSORT. Unfortunately, converting the program to work with BASIC 4.0 requires a lot of work. But, this difficulty can be overcome with the approach presented here.

The problem is that strings are stored differently under BASIC 4.0 than under earlier BASICs. At the end of a string there is a backward pointer which is used to speed up garbage collect. KEYSORT doesn't move strings around, it just changes the array pointers. Consequently, the backward pointers are incorrect. If the PET tries to do a garbage collect, it might crash. You could change the KEYSORT program to fix up the pointers, but you'll find that modifying the BASIC program is easier.

Listing 1 is a simple demonstration of using KEYSORT on a new PET. The steps in this example are:

1. Read in the file to be sorted.
2. Compress strings to the top of memory.
3. Seal off the strings.
4. Invoke the sort.
5. Write out the sorted file.
6. Reset memory pointers.

My version of KEYSORT includes element zero in the sort. My PET has normal memory where most PETs have ROM sockets, so I have relocated the sort to this area. The sort includes null elements.

The key to making the sort operate on BASIC 4.0 is in steps 2 and 3 above. The statement, X equals FRE(0), forces all strings to be compressed to the top of memory. They are sealed off by resetting the top of memory at the same place as the bottom of the strings. Therefore, none of these strings are eligible to participate in garbage collection. Then, when the sort is completed, it doesn't matter that the backward pointers are incorrect. Before the program ends, the top of memory is reset to its original value. (Otherwise, you would permanently lose the memory which we sealed off.) Then, to make no errors occur, the variables are cleared.

```
100 REM  KEYSORT WITH BASIC 4.0
110 REM
120 REM  BY GORDON CAMPBELL
130 REM
140 REM
150 REM STEP 1: READ IN THE FILE TO SORT
160 REM
170 DIM S$(1000): REM ARRAY TO SORT
180 OPEN 8,8,8,"FILE": REM FILE TO READ
190 INPUT#8,S$(N)
200 EF = ST: REM END OF FILE"
210 N = N + 1
220 IF EF=0 THEN 190: MORE FIELDS
230 CLOSE 8
240 REM
250 REM
260 REM STEP 2: COMPRESS STRINGS
270 REM
280 X = FRE(0)
290 REM
300 REM
310 REM STEP 3: SEAL OFF THE STRINGS
320 REM
330 X = PEEK(52): REM SAVE
340 Y = PEEK(53): REM POINTERS
350 POKE 52, PEEK(48): REM SEAL OFF
360 POKE 53, PEEK(49): REM STRINGS
370 REM
380 REM
390 REM STEP 4: INVOKE THE SORT
400 REM
410 SYS10*4096
420 REM
430 REM
440 REM STEP 5: WRITE OUT SORTED FILE
450 REM
460 OPEN 8,8,8, "0:SORTED FILE,S,W"
470 FOR J=0 TO 1000
480 IF S$(J) = "" THEN 500: SKIP NULLS
490 PRINT#8, S$(J)
500 NEXT
510 CLOSE 8
520 REM
530 REM
540 REM STEP 6: RESET MEMORY POINTERS
550 REM
560 POKE 52,X: REM    RESET TOP
570 POKE 53,Y: REM    OF MEMORY
580 CLR: REM   KILL BAD STRINGS
590 END
```

A Disk Menu Program

David C. Oshel, 1219 Harding Ave., Ames, IA 50010

Most of the disk directory management programs I've seen involve flashy READ/WRITE track sector subroutines which deal with disk files directly. The costs are high, both in terms of purchasing, and in comprehension and utility.

The twenty lines of Applesoft which follow provide an elegant and powerful alternative.

The program creates a dynamically-numbered Disk Menu which is susceptible to immediate insertion, deletion or rearrangement of any of the displayed menu items. You may arrange your programs at will, in alphabetical or logical groupings. Further, you may display a program *description* rather than the terse and sometimes cryptic catalog title under which the program actually runs. Moreover, if your catalog contains a suite of programs which call each other, you only need to display the primary program of the system, cutting your menu length by three-fourths in some cases. While your *catalog* may show confusion, your *menu* is logical, orderly and concise!

Menu entries take the form of paired items in DATA statements at the end of the program. There is no limit on the number of items which may be included, subject to disk space. An "empty" menu is valid, while long menus are displayed over subsequent screen pages, each dynamically-numbered.

The first item in each DATA pair is the program's menu description. The second item in the DATA pair is the program's catalog name, the name under which it runs. There must be two entries for each program in the menu, even if one of them is null (that is, " "). Also, some care must be taken when entering DATA statements to

enclose each item in quotes; a missing quote will truncate a lengthy menu, or produce other peculiar effects. Including an initial period in the catalog-name DATA field will suppress DOS, and allows the corresponding *menu* description to be a non-executing documentation line!

RUN, BRUN and EXEC are valid, with two items of interest. First, a machine language utility program which initializes and returns (normally) to BASIC, will initialize and return to the menu *via* the "File Check" statement. You must then exit the program to get at your utility program. Secondly, EXEC will "pend," as it was designed to do, until the program again calls the keyboard for input, at which time the EXEC will commence. The delay is disconcerting the first time you see it, and the "File Check" message in this case is also normal.

One warning: You may dislike the POKE 1012,0 statement in line 20. This causes the disk to reboot whenever RESET is hit, returning you to your HELLO program. In my case, all my disks include this menu program as the HELLO program. I like the "turnkey" feel of a disk containing many otherwise autonomous programs. The POKE may be deleted with no ill effects.

The POKE 216,0 in line 130 re-enables normal error messages before exiting the program.

```
5  REM
     A DISK MENU PROGRAM          BY DAVE OSHEL

10 D$ =  CHR$ (4)
20  PRINT D$"NOMONICO": POKE 1012,0: DIM P$(15,2)
30 PG% = I:SW% = 0: ONERR  GOTO 80
40  TEXT : HOME : PRINT "DISK MENU     ]OOI[      APRIL 27, 1981": PRINT
50  PRINT "PAGE "PG%: INVERSE : VTAB 23: HTAB 10: PRINT "RETURN FOR NEXT PAGE": NORMAL : VTAB 5
60  FOR I = 1 TO 14: IF I < 10 THEN  PRINT " ";
70  READ P$(I,0): READ P$(I,1): PRINT I" - "P$(I,0): NEXT 1: GOTO 90
80 SW% = 1
90  PRINT I" - EXIT THIS MENU":PG% = PG% + I: PRINT
100  PRINT "WHICH, PLEASE? (1-"I;: INPUT ") ";AN$:AN% =  INT ( VAL (AN$)): IF (AN% < 1 OR AN% > 1) AND AN$ <  > "" THEN  VTAB
     ( PEEK (37)): CALL  - 868: GOTO 100
110  IF AN$ = "" AND SW% = 1 THEN  RESTORE : GOTO 30
120  IF AN$ = "" THEN 40
130  IF AN% = I THEN  VTAB 23: CALL  - 958: POKE 216,0: END
140  ONERR  GOTO 160
150  PRINT D$"RUN"P$(AN%,1)
160  ONERR  GOTO 180
170  PRINT D$"BRUN"P$(AN%,1)
180  ONERR  GOTO 200
190  PRINT D$"EXEC"P$(AN%,1)
200  PRINT : CALL  - 958: PRINT "FILE CHECK => "P$(AN%,1): FOR I = 1 TO 2500: NEXT : RESTORE : GOTO 30
1000  REM  ILLUSTRATIVE DUMMY                MENU
1010  DATA "EXAMPLE 1: PROGRAM NOT ON DISK","CANTERBURY TALES"
1020  DATA "EXAMPLE 2: THIS PROGRAM","DISK MENU PROGRAM"
1030  DATA "* EX 3: HOUSEKEEPING MEMOS FORMAT",".PERIOD SUPPRESSED DOS"
```

Auto-Run-Wedge for the PET

Werner Kolbe, Hardstr. 77, CH 5432
Neuenhof, Switzerland

Recently I bought a CBM 3040 floppy disk. I do not have BASIC 4.0 because the ROMs do not fit into my old PET's hardware. Therefore, my first keystrokes after switching on are always the same:

```
LOAD"*",8
```

to load the DOS, support the "wedge" from disk and

```
RUN
```

I wanted to save some of the work — at least the RUN. I discovered the following trick from a program I had analyzed.

The machine code of the wedge consists of two parts: the first one is a jump instruction which is put into the CHR GET routine of PET's BASIC located at $70. The second part loads into the high RAM memory (for example from $7E52 to $8000 for a 32K PET).

What you have to do is create a program file that starts loading at $70 then makes a gap and continues to load into the upper part of the memory. The first part is easy. You open a program file and the first two bytes you write on it are the start address. In your case, this will be CHR$(112) and CHR$(0), which is $0070. The following bytes that you print onto the file are the "program." You have to enter the jump instruction and continue with PET's standard zero page setup until you come to $FB,FC, which is the "load pointer." By changing this pointer you get the necessary gap.

For example, if you put $7D into $FC, the following bytes will be loaded at the location $7DFD and beyond it. Note that it is not possible to change the low part of the load pointer also. In a first attempt, I tried to put $7E52 into it. PET crashed because after loading $52 into $FB the load was continued at $0052, and $FC remained unchanged.

Here is how it is done:

1. Step: Reset the PET, switch it off and on again.
2. Step: Load your wedge into PET and run it.

3. Step: Type in the Shifter program, listing 1, run it and save it. (You may need it again if you make a mistake in one of the following steps).

4. Step: Jump into the resident monitor and enter the bytes listed in listing 2. Save them with the monitor for the same reason as above. (S"1:BYTES",08,056E,0642).

5. Step: Type in the Wedge-Saver program, listing 3, save it, put an empty formatted disk into drive 1 and run it.

Now your "wedge" is ready. Test it! Reset the PET, put the disk into drive 0 and enter LOAD "*",8. Then enter> $, and see that it works. If it doesn't, repeat the steps above and check every byte carefully.

You might think this is a lot of work just to save a RUN, but if you have prepared your wedge file once, you may copy it on all your disks like any other program file.

In the Shifter and in the Saver there are several PEEKs and POKEs, which are necessary because of the various PETs' memory sizes and wedge versions. Now the whole process should work independently of that, and the wedge will load into the upper end automatically.

***Listing 1:* Shifter**

```
10 A=PEEK(52)+256*PEEK(53)+1
15 B=PEEK(52)+6*256+1:C=(PEEK(53)+2)*256-A
20 FORI=0TOC:POKE(I+B),PEEK(I+A):NEXT
```

***Listing 2:* Enter with MLM**

```
C*   PC   SR AC XR YR SP NV*BDIZC
.;  C6FB 33 00 00 00 FE 00110011
.
           0  1  2  3  4  5  6  7
.:  056E 70 00 4C FD 7D 02 E6 78
.:  0576 AD 09 02 C9 3A B0 0A C9
.:  057E 20 F0 EF 38 E9 30 38 E9
.:  0586 D0 60 80 4F C7 52 00 00
.:  058E 20 C1 2E E6 19 90 E0 96
.:  0596 00 1B 00 2A 01 FF 00 00
.:  059E 00 00 00 11 00 18 10 3F
.:  05A6 1B 01 03 20 00 00 27 00
.:  05AE 00 00 03 00 00 5E 00 00
.:  05B6 00 00 00 00 03 00 00 00
.:  05BE 00 00 00 00 00 00 C0 83
.:  05C6 11 00 84 AD AF 00 00 00
.:  05CE 00 00 00 01 0E 60 08 27
.:  05D6 00 00 18 43 01 02 00 00
.:  05DE 00 00 80 80 80 80 80 80
.:  05E6 80 81 81 81 81 81 81 82
.:  05EE 82 82 82 82 82 82 83 83
.:  05F6 83 83 83 00 00 FB 7D A2
.:  05FE 52 86 71 CA 86 34 E6 72
.:  0606 A5 72 85 35 A9 01 85 28
.:  060E A9 04 85 29 A9 00 A8 91
.:  0616 28 C8 91 28 A5 28 18 69
.:  061E 02 85 2A A5 29 69 00 85
.
.:  0626 2B A5 34 A4 35 85 30 84
.:  062E 31 A5 2A A4 2B 85 2C 84
.:  0636 2D 85 2E 84 2F 20 30 C7
.:  063E 6C 71 00 00 AA AA AA AA
```

***Listing 3:* Wedge-Saver**

```
10 POKE43,8:POKE42,0:CLR:OPEN2,8,3,"1:WIDGE,P,W"
15 POKE1394,PEEK(53)-1:POKE1532,PEEK(53)-1
20 POKE1534,PEEK(52)+1
30 FORI=1390TO2049:PRINT#2,CHR$(PEEK(I));:NEXT:CLOSE2
```

MICRO Dealers

Dealer Update January, 1982

Presented here in zip-code order are those MICRO dealers who responded to our newsletter request for information concerning their dealership. Many have been MICRO dealers for quite some time while others are new. This service is provided to acquaint readers with these dealers and to encourage readers to visit dealers in their area. This listing is provided twice a year to update previous listings (see MICRO 35:51). This is not intended as a complete listing of MICRO dealers.

United States

New Jersey

The Computer Forum
80 Broad St.
Red Bank, New Jersey 07701
Contact: Steve Morse
201/530-9103
Services: Courses/Seminars
Hardware: Apple, Atari, CBM, PET
Software: Business, Personal, Educational, Games

New York

The Computer Corner
200 Hamilton Ave.
White Plains, New York 10601
914/WHY-DATA
Services: Seminars, Service Dept.
Hardware: Apple
Software: Business, Personal, Educational, Games, Other

Computer Microsystems
1196 Northern Blvd.
Manhasset, New York 11030
Contact: Andrei Rozwadowski
516/627-3640
Hardware: Apple, CBM, PET, Northstar, Micromation
Software: Business, Personal, Educational, Games

Computer Shop
207 Boices Lane
Kingston, New York 12401
Contact: Clemens Haneke
914/336-8411
Hardware: Apple, Atari, Compustar, Intertec
Software: Business, Personal, Educational, Games

Computerland/Ithaca
225 Elmira Rd.
Ithaca, New York 14850
Contact: Ben Herrmann
607/277-4888
Services: User's Software Library
Hardware: Apple, IBM 8088, Xerox 280
Software: Business, Personal, Educational, Games

Pennsylvania

Computer Store of Pittsburgh
612 Smithfield St.
Pittsburgh, Pennsylvania 15222
Contact: Art Vaughan
412/391-8050
Hardware: Apple, Atari, Northstar, Zenith, Dynabyte, HP
Software: Business, Personal, Educational, Games

The Computer Spot
On the Diamond
Ligonier, Pennsylvania 15658
Contact: Harold Newell
412/238-2381
Hardware: Apple
Software: Business, Personal, Educational, Games

Computers Unlimited
2813 East Prospect Road
York, Pennsylvania 17402
Contact: Keith C. Kirn
717/755-1045
Hardware: Apple, CBM, PET
Software: Business, Personal, Educational, Games

Maryland

The Computer Forum
569 Baltimore-Annapolis Blvd.
Severna Park, Maryland 21146
Contact: Virginia Stibolt
301/544-0909
Services: Education, Rental
Hardware: Atari, CBM, PET, Osborne
Software: Business, Personal, Educational, Games (software support for Apple, TRS-80, Osborne)

Computers Unlimited
907 York Road
Towson, Maryland 21204
Contact: Philip Lester
301/321-1553
Hardware: Apple, CBM, PET
Software: Business, Personal, Educational, Games

Virginia

Community Computers
2704 N. Pershing Drive
Arlington, Virginia 22201
Contact: J. Michael Versace
703/527-4600
Hardware: OSI
Software: Business, Personal, Educational, Games

North Carolina

Byte Shop
218 N. Elm St.
Greensboro, North Carolina 27401
Contact: Bob Terrell
919/275-2983
Services: Service Center for Apple, Epson, Commodore
Hardware: Apple, Atari, PET, Microtek, Epson, Mountain Computer
Software: Business, Personal, Educational, Games

Georgia

Atlanta Computer Mart
5091 Buford Highway
Atlanta, Georgia 30340
404/455-0647
Hardware: Apple, Atari, Northstar, California Computer Systems, Ithaca Intersystems
Software: Business, Personal, Educational, Games

Florida

SEB Computers
1705 University Blvd. N.
Jacksonville, Florida 32211
Contact: Sam Bateh
904/743-7050
Services: Consulting
Hardware: Apple, Atari, CBM, PET
Software: Business, Personal, Educational, Games, Custom Programming

Al Personal Computers
178 Oxford Rd.
Fern Park, Florida 32730
Contact: Pete Bender
305/339-8914
Services: Atari Service Center
Hardware: Apple, Atari, CBM, VIC, TRS-80, Eagle, Xerox
Software: Business, Personal, Educational, Games

Computer Store
2620 East Robinson Street
Orlando, Florida 32803
Contact: Jim Lewis
305/894-8357
Services: Custom Programming, Vocational Programming Classes
Hardware: Apple, Atari, CBM, TRS-80, IBM, Vector Graphic
Software: Business, Personal, Educational, Games, Scientific

Associated Information Systems
825 Osceola Dr.
Rockledge, Florida 32955
Contact: David R. Hendricks
305/632-1090
Services: Computer Service Bureau
Hardware: OSI, Epson, Amdex, Hazeltine, ATV, D.P. Supplies
Software: Business, Personal, Educational, Games, Custom Programming

Michigan

Computer Center
28251 Ford Rd.
Garden City, Michigan 48135
313/425-2470
Hardware: Apple, Atari
Software: Business, Personal, Educational, Games

New Dimensions In Computing Inc.
541 E. Grand River Ave.
E. Lansing, Michigan 48823
Contact: Robert Gibbs
Claus Buchholz
517/337-2880
Services: Repair
Hardware: Atari, Vector Graphic Z80, Exidy Sorcerer Z80, Paper Tiger and Epson Printers
Software: Business, Personal, Educational, Games, Books

Iowa

Cyberia Inc.
2330 Lincoln Way
Ames, Iowa 50010
Contact: Dolores Keith-Taylor
515/292-7634
Services: Software Development, Service Department, Consultation
Hardware: Apple, CBM, PET
Software: Business, Personal, Educational, Games, Agricultural

Computerland
3271 Armar Dr.
Marion, Iowa 52302
Contact: Frank Malone
319/373-1241
Services: Seminars, Training (in-store)
Hardware: Apple, Atari, CBM, Xerox, SAM, 8088, IBM Personal Computer, Vector Graphics
Software: Business, Personal, Educational, Games

Wisconsin

Colortron Computer Division
2111 Lathrop Ave.
Racine, Wisconsin 53405
Contact: Lance Evans
414/637-2003
Services: Custom Programming and Consulting
Hardware: Apple, CBM, PET
Software: Business, Personal, Educational, Games

Century Computer/Phone Concepts
117 South 4th St.
La Crosse, Wisconsin 54601
Contact: Dan Saugstad
608/785-2010
Services: Custom Programming, Bulletin Board Service (1982)
Hardware: Apple, Atari
Software: Business, Personal, Educational, Games

(Continued on page 81)

Dealers *(Continued from page 79)*

Minnesota

Digital Designs
829 3rd Ave. S.E. #265
Rochester, Minnesota 55901
Contact: Steve Surprenant
507/282-3222
Hardware: OSI, Altos (Z80)
Software: Business, Personal, Word Processing, FARM Accounting

Illinois

DigitalWorld
711 Army Trail Road
Addison, Illinois 60101
312/628-9222
Services: Training, Service
Hardware: Atari, CBM, PET, Xerox, Zenith, NEC
Software: Business, Personal, Educational, Games

Oak Brook Computer Centre
17 W. 426 22nd Street
Oakbrook Terrace, Illinois 60181
Contact: Bill Colsher
312/941-9005
Hardware: Apple, Atari
Software: BPI, Personal Software, PFS, etc., (largely business oriented)

Data Domain of Schaumburg
1612 E. Algonquin Rd.
Schaumburg, Illinois 60195
312/397-8700
Services: Business Seminars, Service Dept.
Hardware: Apple, Archives, Alpha Micro, Hewlett-Packard Calculators
Software: Business, Personal, Educational, Games, Large Selection of Books and Magazines

Farnsworth Computer Center
1891 N. Farnsworth Ave.
Aurora, Illinois 60505
Contact: Eleanor Snyder
312/851-3888
Hardware: Apple, Hewlett-Packard
Software: Business, Personal, Educational, Games

The Computer Store of Rockford
3515 Auburn Street
Rockford, Illinois 61103
815/962-7580
Hardware: Apple, Cromemco, Data General, Northstar
Software: Business, Personal, Educational, Games

Missouri

Computer Mart
1904 E. Meadowmere
Springfield, Missouri 65804
Contact: Rob Lurvey
417/862-6500
Services: Consulting
Hardware: Apple, IMS International
Software: Business, Personal, Educational, Games

Kansas

Personal Computer Center
3819 W. 95th Street
Overland Park, Kansas 66206
Contact: Charles Meyers
913/649-5941
Services: Repair Department, Seminars and Training
Hardware: Apple, CBM
Software: Businesss, Personal, Educational, Games

Mr. Computer
1424 Industrial-Emporia W. Plaza
Emporia, Kansas 66801
Contact: Bill or Sue
316/342-4893
Services: Apple and Atari Service Center, Classes and Seminars
Hardware: Apple, Atari
Software: Business, Personal, Educational, Games

Amateur Radio Equipment Co.
1203 E. Douglas
Wichita, Kansas 67211
Contact: Alden Lansdowne
316/264-9166
Services: Amateur, Experimenter
Software: Business, Personal, Educational, Games

Nebraska

MicroAge® Computer Store
4217 South 84th St.
Omaha, Nebraska 68127
Contact: Dennis Hendrickson
402/339-7441
Services: Consulting
Hardware: Apple, Atari, CBM, Altos, Northstar, Archives
Software: Business, Personal, Educational, Games

Arkansas

Computers Etc.
7624 Baseline Rd.
Little Rock, Arkansas 72209
Contact Jerry or Sherry Baldwin
501/562-3200
Hardware: Apple, Z-80 NEC
Software: Business, Personal, Educational, Games, Utility

Texas

Computer Concepts
4699 Calder
Beaumont, Texas 77707
Contact Ed or May Culotta
713/892-3992
Hardware: Apple, Epson Printer
Software: Business, Personal, Educational, Games

Computer 'n Things
2825 Hancock Dr.
Austin, Texas 78731
Contact: Gary Bellamy
512/453-5970
Hardware: Apple, NEC, Cromemco
Software: Business, Personal, Educational, Games

Arizona

Personal Computer Place
1840 W. Southern
Mesa, Arizona 85202
Contact: Roger L. Smith
602/833-8949
Services: Disk Drive Repairs, Program EPROMs
Hardware: Apple, CBM, PET, Southwest Technical Products
Software: Business, Personal, Educational, Games

Nevada

JBA Computers
3111 S. Valley View Ste I-101
Las Vegas, Nevada 89102
Contact: W. Street
702/871-2999
Services: Repair and Maintenance, Contracts, Custom Programming, Circuit Design, Consulting
Hardware: Atari, OSI
Software: Business, Personal, Educational, Games

California

Main-Line Computers
1844 14th Street
Santa Monica, California 90404
Contact: W. Niles
213/450-9924
Hardware: Apple, Atari, OSI
Software: Business, Personal, Educational, Games

Computer Forum
14052 E. Firestone Blvd.
Santa Fe Springs, California 90670
Contact: Dave Hayhurst
213/921-2111
Services: Classes, Service Department, Custom Programming
Hardware: Apple, SYM
Software: Business, Personal, Educational, Games

(Continued)

Dealers *(Continued)*

Omni Unlimited
105 So. Los Robles
Pasadena, California 91101
Contact: Don K. Fenstermaker
213/795-6664
Services: Service Center
Hardware: PET, Zenith
Software: Business, Personal, Educational, Games, Legal, Medical — Database

The Computer Merchant
1040 E. Valley Pkwy.
Escondido, California 92025
Contact: Michael Houlberg
714/746-0662
Hardware: Apple, Xerox
Software: Business, Personal, Educational, Games

Computerland
289 E. Highland Ave.
San Bernardino, California 92404
714/886-6838
Services: Repair, Software Training
Hardware: Apple, Atari, IBM
Software: Business, Personal, Educational, Games

Computer Store of San Leandro
701 MacArthur Blvd.
San Leandro, California 94577
Contact: Hector
415/569-4174
Services: Custom Software, System Design
Hardware: Atari, Altos
Software: Business, Personal, Educational, Games, CPM Software

Byte Shop
6041 Greenback Lane
Citrus Heights, California 95610
916/969-2983 or 5259
Hardware: Apple, Osborne, Northstar, Cromemco
Software: Business, Personal, Educational, Games

Computerland of Sacramento
1537 Howe Ave. #106
Sacramento, California 95825
Contact: Ralph or Gail Lovelace
916/920-8981
Services: Computer Training Seminars
Hardware: Apple, Vector, IBM, Altos
Software: Business, Personal, Educational, Games

Washington

Personal Computers, Inc.
South 104 Freya #104
Spokane, Washington 99202
509/534-3955
Services: Custom Design Hardware and Software; Complete Support for Business Installations
Hardware: Apple, PET
Software: Business, Personal, Educational, Games

Alaska

BG Systems
204 East International
Anchorage, Alaska 99502
Contact: Robert DeLoach
907/276-2986
Services: Client Accounting
Hardware: CBM, PET, Super PET, VIC-20
Software: Business, Personal, Educational, Games

Juneau Electronics
1000 Harbor Way
Juneau, Alaska 99801
Contact: Stan Belzak
907/586-2260
Hardware: Apple
Software: Business, Personal, Educational, Games

MICRO

A NEW STAR ON THE HORIZON

SpeeD★Star

The Applesoft Compiler from Southwestern Data Systems

…tar dramatically increases th… operating speed of your Applesoft programs. It works simply, quickly and automatically — and it offers these important features:

- Compiles et approximately 1200 lines per minute – many times faster than most other Apple compilers since the disk is not accessed during compilation
- Full support for DOS commends, Error Messages, and Graphics
- Special "Compile to Disk" mode allows compilation of programs too large to be done in memory
- Unique selective chaining option allows you to pass just the variables you want to subsequent programs. Because SpeedStar converts Applesoft programs to binary files, you can also have any number of programs in memory at once. Compiled programs can also be used via CALLs as subroutines in ordinary Applesoft programs.
- Unlimited user back-up copies

GET MO E THAN YOU PA OR

When you buy SpeedStar from SDS, you get more than just a great Apple compiler – you get e team of utility software specielists ready to help you solve your software problems. The SDS name is your assurence of quality and service — before and after the sale.

Each SpeedStar package includes a program diskette with complete documentation enclosed in an attractive padded notebook. Each manual is fully indexed with tab dividers for quick and easy reference. Easy-to-follow examples will make you a SpeedStar pro in no time.

REQUIREMENTS

A 48K Apple II with Applesoft in ROM or equivalent. Please specify DOS 3.2 or 3.3.

$134.95

SpeedStar is at your local dealer NOW. If he's out of stock, have him give us a call and we'll ship your copy the same day.

SpeedStar . . . ask for it by name.

SDS
southwestern data systems

P.O. Box 582-M, Santee, CA 92071 • 714-562-3670

* Apple II and Applesoft Basic are trademarks of Apple Computer, Inc.

Pascal Tutorial

Part 3

Victor R. Fricke
325 Ramapo Valley Road
Mahwah, New Jersey 07430

In the previous articles in this series we concentrated on the use of the Apple Pascal operating system software, more than on Pascal itself. This article concentrates on the Pascal language and gives examples of why I think it is easier to program in Pascal than in BASIC. We will discuss disk I/O operations, and compare BASIC control structures, such as IF and GOSUB, to their Pascal equivalents.

One of the Pascal structured data types is the file, a sequence of records. The usual analogy is a file cabinet, which is a sequential collection of folders, each one containing a record. In each folder there may be several different pieces of paper, but they are all related to each other in some way. Each piece of paper is an element of the record, and it can contain several data fields.

For example, a medium-sized business could have its "PERSONNEL" file in one cabinet. Each employee is represented by a folder containing his record. For each record folder, there might be a personal data card, a W-4 form, and a work log.

Each of these "elements" of the record can contain several data fields. The personal data card, for example, would have name, address, home phone, department, date hired, current pay rate, etc.

Just as the accountant does not have enough space on his desk for all this information at once, so also the computer does not have enough central memory for its files. In both cases, the answer is to keep the file in external storage, and to process one record at a time. Disk I/O is the process of selecting one record at a time and bringing it into central memory from the external storage medium for processing.

Pascal files are sequential. Each record is stored in sequence, and retrieved in the same sequence. In this way, Pascal files are like tape files: a new record can only be added at the end of the file.

There is another resemblance to tape files. A Pascal file can only be in one mode at a time; either it is in a read mode, or a write mode — never both at the same time. This, of course, is also the restriction on BASIC disk files.

Creating a File

Following our analogy a little further, suppose our hypothetical accountant were establishing the personnel file for the first time at a new business. Part of his job is to define the internal structure of the file; i.e., what data will be in each person's file. Each record can be as simple or complex as necessary. However, each record will have the same format. In Pascal, the TYPE declaration is used for this purpose.

Figure 1

```
TYPE
  DATE=
     RECORD
         DAY            :  1..31;
         MONTH          :  PACKED ARRAY [1..3]  OF CHAR;
         YEAR           :   0..99
     END; (* DATE *)

  PERSONALDATA=
     RECORD
         LASTNAME       :  PACKED ARRAY  [1..15]  OF CHAR;
         FIRSTNAME      :  PACKED ARRAY  [1..12]  OF CHAR;
         EMPLOYEENO     :  INTEGER;
         DEPARTMENT     :  INTEGER;
         HOMEPHONE      :  PACKED ARRAY  [1..10]  OF CHAR;
         STREETAD       :  PACKED ARRAY  [1..12]  OF CHAR;
         ZIP            :  INTEGER;
         DATEHIRED      :  DATE;
         PAYRATE        :  REAL
     END; (* PERSONALDATA *)

  W4=
     RECORD
         MARRIED        :  BOOLEAN;
         DEPENDENTS     :  INTEGER;
         EXTRAWH        :  REAL
     END; (* W4 *)

  PERSON=
     RECORD
         DATA           :  PERSONALDATA;
         WITHHOLD       :  W4
         REGHOURS       :  ARRAY  [1..52]  OF REAL;
         OTHOURS        :  ARRAY  [1..52]  OF REAL
     END (* PERSON *)

VAR
  WORKER :  FILE OF PERSON
```

Using our personnel file example, the hypothetical accountant might make the Pascal declarations in figure 1.

This results in a file called WORKER, composed of a fairly complex structure of nested records. The beauty of Pascal is that you don't need to worry about memory allocation or how the data is packed into disk files. Pascal takes care of all that automatically.

There can be many more records in a file than can fit into available memory. Fortunately, Pascal deals with a file one record at a time. The way the system deals with an external file is to set up something called a "file buffer variable." This variable serves as a window to peer into the file and examine one record at a time. For this reason, the file buffer variable is frequently referred to as a "window variable."

For a file called WORKER, the file buffer variable is referred to as WORKER^. Each of its components can be individually referred to; for example WORKER^.DATA.LASTNAME, or WORKER^.WITHHOLD.MARRIED.

After the file is defined, it is opened for writing by the REWRITE statement. We can select the same file name or a different one for the disk directory. For clarity, we will call it DISKFILE. To create the file, we use

```
REWRITE(WORKER,'DISKFILE')
```

I like to think of the REWRITE statement as analogous to erasing and rewinding a tape. It does the following:

1. Allocates variable storage space in memory for the defined data structure of the WORKER file.
2. Sets up the file buffer variable WORKER^
3. Places the file name DISKFILE temporarily on the disk directory.
4. Sets the file position pointer to 0 (first record in file).
5. Sets EOF to true (the beginning of the file is also the end of the file if no records exist yet).

EOF is a predefined procedure which returns a Boolean value of TRUE when the file position pointer is at or beyond the end of the file, and FALSE when it is at a record before the end of the file.

The directory entry for the file is only temporary; it will be removed from the directory if the program terminates without executing a

```
CLOSE(WORKER,LOCK)
```

statement. This statement instructs the system to make the temporary diskfile permanent.

Putting Data Into a File

Once your program has established values for the various elements in WORKER^, you can write the resulting record to the file by using

```
PUT(WORKER)
```

This statement puts the record into the file buffer area of memory and then advances the file position pointer to the start of the next record. Depending on size, it may also cause the system to write the buffer block onto the disk. It doesn't always happen with each PUT, because the block size is always 512 bytes, but the record can be any size.

Reading From a File

To extract data from a file, you have to open it for reading. The RESET statement does this. For our example, it would be

```
RESET(WORKER,'DISKFILE')
```

If you think of the tape analogy, this statement is like rewinding the tape and reading the first record into the window variable. If you RESET a file that is already open, you will make the window variable have the values associated with the first record in the file. In a way, this is like

```
10 RESTORE
20 READ
```

in BASIC.

EOF is left with the value of FALSE by a RESET. Once the file is RESET, to read subsequent records use the GET statement. GET(WORKER) reads the record at the current file pointer location and then advances the file position pointer.

Updating a File

In any file that requires sequential access as described in this article, it would be awkward to update any record if the sequential access were strictly observed. You would have to read and write back all the records before the one you wanted to update. It would be far more convenient to go directly to the record you want, and write its new value.

The SEEK statement allows you to do this. Each record has a sequence number associated with it (the first record is number 0). By doing a SEEK and then a PUT, you can update a record.

The SEEK statement looks like this:

```
SEEK(WORKER,57)
```

This statement advances the file position pointer so that it is at the beginning of the 58th record (record 57). The next GET or PUT will then operate on the desired record.

Closing a File

After you finish reading or writing records in the file, one or more records may remain in the file buffer. The directory entry of the file name is temporary.

To preserve the data in the file buffer and to deallocate the file buffer, it is necessary to close the file before the program terminates. This is done by using the

```
CLOSE(WORKER,LOCK)
```

statement.

Of course, you may not want to save the data that you have placed in the disk file. In this case,

```
CLOSE(WORKER,PURGE)
```

will delete the temporary directory entry created by the REWRITE(WORKER, 'DISKFILE') statement. Since the directory no longer contains the file name 'DISKFILE', it considers the space formerly occupied by the temporary file as <UNUSED>.

IORESULT

In many applications, it would be a good idea to test for errors when a disk I/O operation is attempted. The disk could be bad, the wrong volume might be on line, or the file you want to update may not yet exist.

BASIC provides the ON ERR GOTO statement so that you can recover from an error without an abnormal termination of the program operation. Pascal provides a similar capability with the predefined IORESULT function.

To use IORESULT, you have to turn off the I/O error checking by using the compiler directive (*$I - *) first; otherwise, the program will terminate before you can check the value returned by IORESULT.

IORESULT returns an integer value from 0 to 14, as listed in table 2 of Appendix B in the Apple Pascal Language Manual. If the value is 0, there is no I/O error. For the other codes, you may want to define some error recovery procedures, and perhaps use a CASE statement to invoke them.

Putting It All Together

Now that we have covered the highlights of Pascal disk I/O, it will be instructive to look at an example. Listing 1 creates and updates a simplified personnel file. The file structure has been simplified from our previous examples to keep the program short.

There are similarities and differences between Pascal and BASIC. The similarities should make it easy for anyone who knows BASIC to learn Pascal quickly. The differences are what allows you to write better programs more quickly in Pascal.

Variables

In many versions of BASIC, a variable name can only be one letter, or a letter and a digit. Applesoft is slightly better because you can use a variable name of almost any length. Unfortunately, however, only the first two characters count. That is, SUNDOWN, SUDS, and SU are all the same variable in Applesoft.

In Pascal, variable names can also be quite lengthy, but Pascal variables are distinguished by the first eight characters. So the examples given in the previous paragraph would be distinct in Pascal. This allows you to use identifiers that have a better mnemonic relationship to the quantity represented by the variable.

As an example, it is much easier to remember the meanings of INTRATE, INTEREST, and INVOICE than of I, I2, and I4.

In BASIC there are a limited number of variable types; integer (A%), real (A), and string (A$). These variable types are implicitly defined by the suffix character on the variable name.

Listing 1

```
PROGRAM PERSONNEL(WORKER);

TYPE
   PERSON=
      RECORD
         LASTNAME     :STRING[15];
         EMPLOYEENO   :INTEGER;
         MARRIED      :BOOLEAN;
         DEPENDENTS   :INTEGER;
         PAYRATE      :REAL
      END; (* PERSON RECORD *)


VAR
   WORKER     :FILE OF PERSON;
   FILENAME   :STRING;
   NUMBER,
   RUGRATS,
   RECNO      :INTEGER;
   NAME       :STRING[15];
   STATUS     :BOOLEAN;
   RATE       :REAL;


PROCEDURE SHOWENDRECNO;          (* INDICATE LOCATION OF EOF *)
BEGIN
   RECNO := 0;
   RESET(WORKER);
   WHILE NOT EOF(WORKER) DO
      BEGIN
         GET(WORKER);
         RECNO := RECNO + 1;
      END; (* WHILE *)
   WRITELN('EOF IS ',RECNO);
END;

PROCEDURE EMPTY;                 (* CLEAR RESIDUAL JUNK *)
BEGIN
   WITH WORKER^ DO
   BEGIN
      LASTNAME := '               ';(* 15 BLANKS *)
      EMPLOYEENO := 0;
      MARRIED := FALSE;
      DEPENDENTS := 0;
      PAYRATE := 0.0;
   END;
END;


PROCEDURE NAMEIT;
BEGIN
   WRITE('LAST NAME       :');
   READLN(NAME);
   WORKER^.LASTNAME := NAME;
END;



PROCEDURE NUMBERIT;
BEGIN
   WRITE('EMPLOYEE NO.    :');
   READLN(NUMBER);
   WORKER^.EMPLOYEENO := NUMBER;
END;


PROCEDURE MARRYIT;
VAR
   ANSWER  : CHAR;

BEGIN
   WRITE('MARRIED? (Y/N) :');
   READ(ANSWER);
   WRITELN;
   IF ANSWER = 'Y' THEN STATUS := TRUE
                   ELSE STATUS := FALSE;
   WORKER^.MARRIED := STATUS;
END;
```

```
PROCEDURE DEPENDIT:
BEGIN
   WRITE('NUMBER OF RUGRATS    :');
   READLN(RUGRATS);
   WORKER^.DEPENDENTS := RUGRATS:
END;

PROCEDURE PAYIT;
BEGIN
   WRITE('PAY RATE               :$');
   READLN(RATE);
   WORKER^.PAYRATE := RATE;
END;


PROCEDURE FILLOUT;          (* GET VALUES FOR RECORD ELEMENTS *)
BEGIN
   WRITELN('RECORD NO: ',RECNO);
   NAMEIT;
   NUMBERIT;
   MARRYIT;
   DEPENDIT;
   PAYIT;
END:


PROCEDURE APPEND;           (* STICK A RECORD ON THE END OF FILE *)
BEGIN
   EMPTY;
   FILLOUT:
   PUT(WORKER);
END;


PROCEDURE DISPLAY:          (* SHOW THE CURRENT RECORD *)
BEGIN
  PAGE(OUTPUT);
  WITH WORKER^ DO
     BEGIN
       WRITELN('1:LAST NAME          : ',LASTNAME);
       WRITELN('2:EMPLOYEE NO.       : ',EMPLOYEENO);
       WRITE(  '3:MARITAL STATUS     : ');
         CASE MARRIED OF
           FALSE: WRITELN('SINGLE');
           TRUE : WRITELN('MARRIED');
         END; (* CASES *)
       WRITELN('4:# OF DEPENDENTS    : ',DEPENDENTS);
       WRITELN('5:PAY RATE           :$',PAYRATE);
     END (* WITH *):
END: (* DISPLAY *)

PROCEDURE UPDATE;               (* MODIFY A RECORD *)
VAR
   CHOICE  : INTEGER;
   ANSWER  : CHAR;

 BEGIN
    DISPLAY:
    WRITELN('SELECT NUMBER OF ITEM TO BE CHANGED.');
    WRITELN('SELECT 0 TO QUIT.'):
    RESET(WORKER):
    CHOICE := 1;
    WHILE CHOICE >0 DO
       BEGIN
          READLN(CHOICE);
          IF (CHOICE>0) AND (CHOICE<6) THEN
             CASE CHOICE OF
                1: NAMEIT:
                2: NUMBERIT;
                3: MARRYIT;
                4: DEPENDIT;
                5: PAYIT:
             END; (* CASES *)
       END: (* WHILE *)
    DISPLAY;
    WRITELN('IS EVERYTHING OK? '):
    READ(ANSWER);
    WRITELN:
    IF ANSWER <> 'Y' THEN UPDATE
       ELSE
          BEGIN
```

(Continued)

In Pascal, a wide variety of variable types are available. Besides INTEGER, REAL, and STRING, you have other predefined types like CHAR, BOOLEAN, and LONG INTEGER. You can also define your own data types. As a matter of fact, you must declare all variables to be of a specific type before you use them in a Pascal program.

The big advantage of Pascal is that you are not confined to using just these predefined types of data; you can define your own data types, and use variables which have values of those types.

There are lots of advantages to defining your own data types. For instance, you could represent the months of the year by their ordinal integers; e.g., 1 for January, 2 for February, etc. In BASIC this is the only way to do it. But in Pascal you can define a new data type MONTH as follows:

```
TYPE
  MONTH  =  (JAN,FEB,MAR,
  APR,MAY,JUN,JUL,AUG,SEP,
  OCT,NOV,DEC);

VAR
  BIRTHMONTH : MONTH;
```

This method allows you to use an assignment statement in your program which is straightforward and easy to understand:

```
BIRTHMONTH := MAY;
```

Assignments

In BASIC, the symbol "=" has an ambiguous meaning. It can represent the assignment of a value to a variable, as in

```
50 A = 27
```

Or it can represent the Boolean equality operator; i.e.,

```
50 IF A = B THEN 100
```

In the above example, A = B is an expression that is either true or false. I have called this ambiguous because the fragment A=B can be either a logical (Boolean) expression, or an assignment instruction, depending on context. Also, the Boolean value of "false" is the same as the real value of 0 in BASIC, lending further ambiguity.

In Pascal this ambiguity is not present. An assignment statement uses the "assignment operator," ":=", as in the example

```
A := B;
```

```
          SEEK(WORKER,RECNO);
          PUT(WORKER);
       END; (* ELSE *)
END;

PROCEDURE DOIT;                    (* THIS IS THE PROCEDURE     *)
BEGIN                              (* THAT CONTROLS THE OTHERS *)
   PAGE(OUTPUT);
   SHOWENDRECNO;
   WRITELN('RECORD NO? (-1 TO QUIT)');
   READLN(RECNO);
      WHILE RECNO >= 0 DO
        BEGIN
          RESET(WORKER);
          SEEK(WORKER,RECNO);
          GET(WORKER);
          IF EOF(WORKER) THEN APPEND
                         ELSE UPDATE;
          WRITE('RECORD NO? (-1 TO QUIT)');
          READLN(RECNO);
        END; (* WHILE *)
END; (* DOIT *)


PROCEDURE ERRHANDLE;            (* OPENS FILE IF NEW *)
VAR
   ANSWER   : CHAR;

BEGIN
   WRITELN('NO SUCH FILE ON DISK.');
   WRITE('DO YOU WANT TO ADD IT? ');
   READ(ANSWER);

   IF ANSWER = 'Y' THEN
      BEGIN
         REWRITE(WORKER,FILENAME);
         DOIT;
      END; (* IF *)
END; (* ERRHANDLE *)


BEGIN  (* MAIN PROGRAM *)
   WRITE('FILE? ');
   READLN(FILENAME);

(*$I-*)
   RESET(WORKER,FILENAME);
(*$I+*)

   IF IORESULT = 0 THEN DOIT
                   ELSE ERRHANDLE;

   CLOSE(WORKER,LOCK)
END.
```

while a Boolean expression uses the "equality operator," "=". The BASIC statement

100 IF A=B THEN C=D

results in only C changing its value. This becomes more obvious in the equivalent Pascal statement

IF A = B THEN C := D;

IF Statements

In BASIC the IF statement has the format

10 IF expression THEN statement1

20 statement2

If expression is true, then statement 1 is executed; otherwise statement 2 is executed. In Pascal, the structure of the IF statement is exactly the same. It can also be written in this form:

IF expression THEN statement1
ELSE statement2;

Multiple Branches

The situation where your program must choose between multiple alternatives provides an interesting comparison between BASIC and Pascal. In BASIC, such a branch statement might be used to select a subroutine based on student grades, for example. You would need to use a "computed GOSUB" statement:

10 ON GRADE GOSUB 1000, 1750, 2235, 8870, 9025

This type of statement requires GRADE to have an integer value of 1

through 5, which might correspond to a grade of F through A. The difficulty here is that you have to do a lot of page-flipping to see what each of those subroutines does.

In the Pascal version of this selection statement, there are two significant differences. You can make GRADE have values of A, B, C, D, or F by defining your own data type:

```
TYPE
  SCORE = (A,B,C,D,F);
VAR
  GRADE : SCORE;
```

You can also use meaningful procedure names for each of the target procedures, instead of the statement numbers which have no meaning. Thus, the example of multiple branching in Pascal would be:

```
CASE GRADE OF
  A: EXEMPTFROMFINAL;
  B: OPTIONALFINAL;
  C: FINALEXAM;
  D: COUNSEL;
  F: FAIL
END;
```

Looping

Anyone familiar with BASIC knows about FOR...NEXT loops. This structured command is used to repeat a series of instructions a predefined number of times. The FOR statement sets up an index to count the number of iterations, gives the index its initial value, does the block of instructions, increments the index, and finally checks to see if the upper limit of the index has been reached. If it has, the loop is satisfied, and the instruction following the loop is executed. If the index has not yet reached its limit, the block is executed repeatedly until the loop is satisfied.

Pascal also has a FOR statement, which looks like this:

```
FOR I := LOW TO HIGH DO
  BEGIN
    FIRSTSTEP;
    SECONDSTEP;
    THIRDSTEP;
  END;
```

There are several differences between BASIC and Pascal. The STEP clause of BASIC is not available; the index can only be incremented by 1. You cannot count by twos, by fives, or by anything else. You can count down, by using the reserved word DOWNTO in place of TO in the FOR statement, as in

```
FOR I := HIGH DOWNTO
    LOW DO
```

In BASIC, the parameters in the FOR statement may have real or integer values. In the Pascal version, they must be ordinal, such as INTEGER, integer subrange, CHAR, or a user defined type, such as the type MONTH referred to earlier in this article.

If you want to perform other types of loops in BASIC, you have to get very clever with IF statements, indexes, etc. For example, suppose you want to loop for as long as it takes for a process to return a desired result, and you have no way of knowing in advance how many iterations it takes. An example of this type of loop would be the use of the Newton-Raphson method of finding the roots of an equation.

To do this, you need to start with an approximate answer, calculate a new approximate answer, and compare the two results to see how close they are to being the same. This process is repeated until the difference between the two successive answers is acceptably small.

Doing this in BASIC is awkward. In Pascal, it is quite easy. It is done using the REPEAT loop:

```
REPEAT
  STEP1;
  STEP2;
  STEP3;
  STEP4
UNTIL DIFFERENCE < =
  DELTA;
```

Of course, you must be sure that somewhere in the body of this loop DIFFERENCE, it is affected in such a way that it will eventually satisfy the condition in the UNTIL statement. Otherwise, you will have an infinite loop, just as easily as you can in BASIC.

An additional Pascal looping structure not available in BASIC is the WHILE loop:

```
WHILE DIFFERENCE > =
  DELTA DO
  BEGIN
    STEP1;
    STEP2;
    STEP3;
    STEP4
  END;
```

The difference between the REPEAT...UNTIL structure and the WHILE...DO structure is that in the former, the condition for satisfying the loop is evaluated after the loop is executed, while in the latter case the condition is evaluated before executing the loop.

Functions and Procedures

In Pascal terminology a set of instructions that is self-contained and constitutes a subprogram is called a procedure. In BASIC it is called a subroutine. The BASIC subroutine is called into operation by "GOSUB line number." The Pascal procedure is activated by using its name in the main program. As discussed before, this tends to make Pascal programs easier to understand.

Another difference between these languages is parameter passing. In Pascal the values of the variables modified by the procedure are carefully controlled by the use of local variables and formal parameters. In a BASIC program all variables are global; that is, all parts of the program have equal access to each variable. This can result in some hard-to-find bugs when a variable is changed in a remote part of the program while you are expecting it to have a different value.

By keeping all variables as local as possible in Pascal, and passing the values through formal parameters, this problem is eliminated. For a thorough discussion of this area, see reference 1 in the bibliography at the end of this article.

Functions are similar in both languages. In BASIC a function is defined in a DEFFN statement. It, too, has a formal parameter (referred to as its argument in BASIC) like it does in Pascal. In Pascal a function is defined in a FUNCTION subprogram.

Summary

In this series of articles I have given a general overview of the Apple Pascal system, with emphasis on the operating system. The more I got into the Pascal language, the more I was convinced that Pascal has many advantages over BASIC for most applications except for very short programs.

One of my motives for writing this series was the lack of good references for learning Pascal on your own. I wanted to share some of my hard-won

knowledge with others and save them the trouble I had digging up all this information.

In the time since this series was started, several more references have become available. The following bibliography should prove useful to anyone seriously interested in learning Pascal.

Bibliography

1. "Pascal Primer," by David Fox and Mitchell Waite, Howard W. Sams Co., Inc.
2. "Apple Pascal, A Hands-on Approach," by Arthur Luehrmann and Herbert Peckham, McGraw-Hill Book Company.
3. "Beginner's Guide for the UCSD Pascal System," by Kenneth L. Bowles, BYTE Books.
4. "Programming in Pascal," by Heathkit Educational Systems.
5. "Pascal Programming Structures," by George W. Cherry, Reston Publishing Company.
6. "Pascal With Style — Programming Proverbs," by Henry F. Ledgard, John F. Hueras and Paul A. Nagin, Hayden Book Company, Inc.
7. "Apple Pascal Operating System Reference Manual," and "Apple Pascal Language Reference Manual," Apple Computer Inc.
8. "Pascal User Manual and Report," by Kathleen Jensen and Niklaus Wirth, Springer-Verlag.
9. "Programming in Pascal," by Peter Grogono, Addison-Wesley.
10. "Introduction to Pascal, Including UCSD Pascal," by Rodnay Zaks, Sybex.
11. "Microcomputer Problem Solving Using Pascal," by Kenneth L. Bowles, Springer-Verlag.

MICRO

RELOC

RELOC allows the Apple Pascal text editor to be used with DOS 3.3 to more easily edit BASIC text files.

Robert Walker
6100F Wood Chase Lane
Marietta, Georgia 30067

With the recent purchase of Apple Pascal, one shortcoming of Applesoft and Integer BASIC has become evident. Unlike Pascal, BASIC is not supported by a powerful text editor. This article describes how to use the Pascal editor to develop BASIC programs.

With the following program (hereafter known as RELOC) BASIC programs, written with the Pascal editor, are relocated from a Language System text file to a DOS 3.3 exec file. This file is then executed, thus loading the BASIC program into memory for running.

Using the editor for developing BASIC programs offers many advantages. (See listing 1.) Variable names throughout an entire program, for instance, may be changed in seconds. Like Pascal, indentation may be used to illustrate FOR-loop nesting. The feature I find most convenient is the ability to effectively document a program. All text enclosed in brackets is ignored by RELOC and not sent to the exec file, thus eliminating the need for REM statements in the final program. This means the final program requires less memory and executes faster.

Another important feature of RELOC is that all programs are initially relocated into an exec file. This is a handy means of storing often-used subroutines. We may have, for example, a subroutine (saved as an exec file) called "PLOT," which plots data from an array. After completing the main program (with it still in memory) simply "EXEC PLOT" to insert the subroutine into the main program. Provided there is correct interaction of variables, and no overlapping of line numbers, everything should work fine.

Program Implementation

The ability to read Language System disks while operating in DOS 3.3 is made possible by the fact that both systems utilize identical methods to physically format disks. Like DOS 3.3, Language System disks have 35 tracks, numbered 0 through 34. Each track is divided into 16 sectors, numbered 0 through 15. Furthermore, each sector contains 256 bytes. By using the RWTS (read or write a track or sector; see pp. 94-98, *The DOS Manual*) subroutine, data on a disk can be accessed by simply specifying its location in track-sector pairs.

Finding the location of our Language System text file is easy. The Filer command "E(XTENDED LIST" supplies the starting block address and the number of blocks occupied by the text file. Equally convenient is the fact that Language System files are stored in consecutive blocks. If a file starts at block 235 and occupies three blocks, then the file resides in blocks 235, 236, and 237. Simple! Now for the real problem.

How are these blocks associated with track-sector pairs? First, a block is actually two sectors (512 bytes). Each block number represents two track-sector pairs. Second, with some experimenting, I've found that each track contains exactly 8 blocks (i.e., no blocks are split between tracks; see table 1). Blocks 0 through 7 occupy track 0. Blocks 8 through 15 occupy track 1, and so on. Finally, blocks 272 and 279 occupy track 34. In short, the track associated with any block can be computed by the following formula:

TRACK NUMBER = INT(BLOCK NUMBER / 8)

Next, we must determine the two sectors associated with each block. Because these sectors follow no logical

Listing 1

```
                    [ PASCAL-BASIC TEXT FILE TRANSFER ]
                    [       WRITTEN BY R. WALKER      ]
                    [          WICHITA,  KS           ]

 [ INITIALIZE VARIABLES ]

    10  HIMEM:4095  [ PROTECT DISK I/O BUFFER ]
    20  DIM L%(7),H%(7)
    30  D$=CHR$(4)  [ CTRL-D ]
    32  NL%=0    [ NULL ASCII ]
    34  DL%=16   [ DLE ASCII ]
    36  LB%=91   [ LEFT BRACKET ASCII ]
    38  RB%=93   [ RIGHT BRACKET ASCII ]
    40  K=7.99999   [ NUMBER BLOCK PER TRACK
                      SLIGHTLY LESS THAN 8 TO PREVENT ROUNDOFF ERRORS ]

        [ POKE RWTS CONTROL STATEMENTS
          CONTROLLING SUBROUTINE:   768-776
          I/O CONTROL BLOCK:   777-793
          DEVICE CHARACTERISTIC TABLE:   794-797 ]

    50  FOR I = 768 TO 797:  READ C%:  POKE I,C%:  NEXT

        [ LOAD SECTOR-BLOCK ASSOCIATION TABLE ]

    60  FOR I = 0 TO 7:  READ L%(I),H%(I):  NEXT


 [ INTRODUCTION ]

    70  HOME:  VTAB(3)
    80  PRINT "    PASCAL-BASIC TEXT FILE TRANSFER"
    90  PRINT:PRINT
   110  INPUT "SOURCE DRIVE # (PASCAL)-- ";SD%
   120  INPUT "OBJECT DRIVE # (BASIC)-- ";OD%
```

(Continued)

Table 1: Block-Track-Sector Association Table

Block	Track	First Sector	Second Sector
0	0	0	14
1	0	13	12
2	0	11	10
3	0	9	8
4	0	7	6
5	0	5	4
6	0	3	2
7	0	1	15
8	1	0	14
9	1	13	12
10	1	11	10
11	1	9	8
12	1	7	6
13	1	5	4
14	1	3	2
15	1	1	15
-	-	-	-
-	-	-	-
272	34	0	14
273	34	13	12
274	34	11	10
275	34	9	8
276	34	7	6
277	34	5	4
278	34	3	2
279	34	1	15

order, this proves to be more difficult. Table 1 shows that within each track, these sector locations repeat themselves. For example, the first block of any track consists of sector 0, followed by sector 14. This sequence is stored in DATA statements and later read into arrays L% and H%. Locating sector numbers would procede as follows.

Let's assume that some arbitrary block X is the Nth block (numbered 0 through 7, where N = TRACK NUMBER MOD 8) of some arbitrary track. The value L%(N) would be equal to the low (first) sector of block X. Likewise, H%(N) would be equal to the high (second) sector of block X. We now have an algorithm for determining the two track-sector pairs associated with each block number.

As mentioned earlier, the RWTS subroutine is used to read the Language System text file. During program initialization, the RWTS controlling subroutine is stored in locations 768-776. Then, the RWTS IOB (I/O control block) is stored in locations 777-793. And finally, the RWTS device characteristic table is stored in locations 794-797.

During execution of RELOC, five important locations in the IOB are set. First, location 779 contains the drive

Listing 1 *(Continued)*

```
 130  PRINT:PRINT
 140  INPUT "STARTING BLOCK-- ";SB%
 150  INPUT "NUMBER BLOCKS--- ";NB%
 160  PRINT:PRINT
 170  INPUT "EXEC FILE NAME-- ";F$
 180  PRINT:PRINT

[ LOAD DATA BUFFER WITH PASCAL TEXT FILE ]

 190  PRINT "INSERT PASCAL DISK IN DRIVE #";SD%
 200  INPUT "THEN HIT RETURN...";A$
 210  HOME:  PRINT "READING PASCAL TEXT FILE..."
 220  PG%=15 [ PAGE COUNTER:  PG%*256 = BEGIN LOC. OF DATA I/O BUFFER ]
 222  POKE 779,SD%  [ DISK DRIVE NUMBER ]
 224  POKE 789,1  [ COMMAND CODE:  READ ]

 230  FOR BL [BLOCK NUMBER] = SB%+2 [ SKIP 1ST 2 BLOCKS ] TO SB%+NB%-1
 240     TR% = INT(BL/8)  [ COMPUTE TRACK ]
 250     BI% = INT((BL/8-INT(BL/8))*8)  [ RELATIVE BLOCK NUMBER;
                                          BI% = BL MOD 8 ]
 255     PS%=PG%+1

         [ READ FIRST SECTOR OF BLOCK ]

 260     POKE 781,TR%  [ STORE TRACK NUMBER IN IOB ]
 270     POKE 782,L%(BI%)  [ STORE SECTOR NUMBER IN IOB ]
 280     POKE 786,PG%
 290     CALL 768  [ CALL RWTS SUBROUTINE ]

         [ READ SECOND SECTOR OF BLOCK ]

 330     PG%=PG%+1
 340     POKE 782,H%(BI%)
 350     POKE 786,PG%
 360     CALL 768

 370  NEXT

[ CREATE DOS EXEC FILE ]

 380  PRINT:  PRINT "INSERT BASIC DISK IN DRIVE #";OD%
 390  INPUT "THEN HIT RETURN...";A$
 400  HOME:  PRINT "CREATING DOS EXEC FILE..."
 410  PRINT D$;"OPEN";F$;",D";OD%
 420  PRINT D$;"WRITE";F$
 430  FL%=0  [ REMARKS FLAG ]
 440  FOR X = 4098 [ FIRST TEXT CHAR ] TO 4095+(NB%-2)*512 [ LAST CHAR ]
 442     IF PEEK(X)=DL% GOTO 520 [ SKIP DLE ]
 444     IF PEEK(X-1)=DL% GOTO 520 [ SKIP INDENTATION CODE ]
 450     IF PEEK(X)=LB% [ L. BRACKET ] THEN FL%=1:  GOTO 520  [ IGNORE TEXT ]
 460     IF PEEK(X)=RB% [ R. BRACKET ] THEN FL%=0:  GOTO 520  [ SAVE TEXT ]
 470     IF FL%=1 GOTO 520  [ IGNORE TEXT ]
 480     IF PEEK(X)=NL% GOTO 520  [ SKIP NULL ]
 510     PRINT CHR$(PEEK(X));  [ SEND CHAR TO EXEC FILE ]
 520  NEXT
 525  PRINT  [ MAKE SURE ALL CHARS SENT ]
 530  PRINT D$;"CLOSE";F$
 540  PRINT "FILE TRANSFERED"
 550  INPUT "ANOTHER (Y/N)";A$
 560  IF LEFT$(A$,1)="Y" GOTO 70
 999  END

[ RWTS CONTROLLING SUBROUTINE CODE ]

 1000 DATA 169,3,160,9,32,217,3,96,0

[ RWTS I/O CONTROL BLOCK ]

 1100 DATA 1,96,1,0,0 [TRK],0 [SEC],26,3,0,16,0,0,1 [READ],0,254,96,1

[ RWTS DEVICE CHARACTERISTIC TABLE ]

 1200 DATA 0,1,239,216

[ SECTOR-BLOCK ASSOCIATION TABLE ]

           [ SECTORS              RELATIVE    ]
           [ LOW,HIGH           BLOCK NUMBER  ]
 1300 DATA    0 , 14               [ 0 ]
 1310 DATA   13 , 12               [ 1 ]
 1320 DATA   11 , 10               [ 2 ]
 1330 DATA    9 ,  8               [ 3 ]
 1340 DATA    7 ,  6               [ 4 ]
 1350 DATA    5 ,  4               [ 5 ]
 1360 DATA    3 ,  2               [ 6 ]
 1370 DATA    1 , 15               [ 7 ]
```

number from which the Language System text file is read. Secondly, locations 781 and 782 contain the track and sector number, respectively. Location 786 contains the high-order byte of the data buffer starting address. The low-order byte is set to zero. Note that RELOC uses memory locations 4096 (page 16 of memory) and higher for the data buffer. The most important location in the IOB is 789. This contains the command code, which tells the RWTS subroutine whether to read or write. It is important to make sure that this code is 1; otherwise, the Language System text file may be destroyed!

The format of Language System text files is straightforward (see pp. 185, *Apple Pascal Reference Manual*). Each text file begins with a two-block header page containing information used by the text editor. As a result, these first two blocks are not read by RELOC. Our text actually begins in the third block. Each line of text is preceded by a DLE (Data Link Escape), followed by a one-

byte indentation code. These two bytes are ignored by RELOC. Each line is then terminated with a carriage return. The end of the text is padded out with null characters.

With the text file in memory, the only thing left is to create the DOS 3.3 exec file. First, RELOC opens the exec file witb the name assigned by the user. Next, each character, witb the exception of control characters and bracketed comments, is sent to the exec file. Finally, the exec file is closed.

Listing 2

```
10  HIMEM: 4095
20  DIM L%(7),H%(7)
30 D$ =  CHR$ (4)
32 NL% = 0
34 DL% = 16
36 LB% = 91
38 RB% = 93
40 K = 7.99999
50  FOR I = 768 TO 797: READ C%: POKE I,C%: NEXT
60  FOR I = 0 TO 7: READ L%(I),H%(I): NEXT
70  HOME : VTAB (3)
80  PRINT "    PASCAL-BASIC TEXT FILE TRANSFER"
90  PRINT : PRINT
110  INPUT "SOURCE DRIVE # (PASCAL)-- ";SD%
120  INPUT "OBJECT DRIVE # (BASIC)--- ";OD%
130  PRINT : PRINT
140  INPUT "STARTING BLOCK-- ";SB%
150  INPUT "NUMBER BLOCKS--- ";NB%
160  PRINT : PRINT
170  INPUT "EXEC FILE NAME-- ";F$
180  PRINT : PRINT
190  PRINT "INSERT PASCAL DISK IN DRIVE #";SD%
200  INPUT "THEN HIT RETURN...";A$
210  HOME : PRINT "READING PASCAL TEXT FILE..."
220 PG% = 15
222  POKE 779,SD%
224  POKE 789,1
230  FOR BL = SB% + 2 TO SB% + NB% - 1
240 TR% =  INT (BL / K)
250 BI% =  INT ((BL / K -  INT (BL / K)) * 8)
255 PG% = PG% + 1
260  POKE 781,TR%
270  POKE 782,L%(BI%)
280  POKE 786,PG%
290  CALL 768
330 PG% = PG% + 1
340  POKE 782,H%(BI%)
350  POKE 786,PG%
360  CALL 768
370  NEXT
380  PRINT : PRINT "INSERT BASIC DISK IN DRIVE #";OD%
390  INPUT "THEN HIT RETURN...";A$
400  HOME : PRINT "CREATING DOS EXEC FILE..."
410  PRINT D$;"OPEN";F$;",D";OD%
420  PRINT D$;"WRITE";F$
430 FL% = 0
440  FOR X = 4098 TO 4095 + (NB% - 2) * 512
442  IF  PEEK (X) = DL% GOTO 520
444  IF  PEEK (X - 1) = DL% GOTO 520
450  IF  PEEK (X) = LB% THEN FL% = 1: GOTO 520
460  IF  PEEK (X) = RB% THEN FL% = 0: GOTO 520
470  IF FL% = 1 GOTO 520
480  IF  PEEK (X) = NL% GOTO 520
510  PRINT  CHR$ ( PEEK (X));
520  NEXT
525  PRINT
530  PRINT D$;"CLOSE";F$
540  PRINT "FILE TRANSFERED"
550  INPUT "ANOTHER (Y/N)";A$
560  IF  LEFT$ (A$,1) = "Y" GOTO 70
999  END
1000  DATA  169,3,160,9,32,217,3,96,0
1100  DATA  1,96,1,0,0 ,0 ,26,3,0,16,0,0,1 ,0,254,96,1
1200  DATA  0,1,239,216
1300  DATA     0 , 14
1310  DATA    13 , 12
1320  DATA    11 , 10
1330  DATA     9 ,  8
1340  DATA     7 ,  6
1350  DATA     5 ,  4
1360  DATA     3 ,  2
1370  DATA     1 , 15
```

Program Operation

Writing a BASIC program using the text editor requires two special considerations. First, no BASIC line statement, including the carriage return, may exceed 80 characters. This limitation is due to the 80-column format of the editor. I haven't found this to be a significant handicap. Second, by enclosing in brackets, comments can be placed anywhere in the program text. This means that no print statements, for instance, may contain brackets. Naturally, it would still be possible to print brackets by using the CHR$ statement.

Once the text is written and saved, enter the Filer and execute the "E)XTENDED LIST" command. Note the beginning block and the number of blocks occupied by the text file. These will be needed later by RELOC, but the Language System is no longer needed. Boot DOS 3.3, and run RELOC.

The first item requested by RELOC is the source drive number. This is the drive from which the Language System text file will be read. Likewise, the object drive number will be requested. This is the drive to which the exec file will be written. When requested, enter the starting block number given by the "E)XTENDED LIST" command. RELOC automatically skips the first two blocks. Therefore, it is unnecessary for the user to add two to the starting block number. The last request made by RELOC is the exec file's name. Any legal DOS 3.3 file name will do.

RELOC will then prompt the user to insert the Pascal disk. Once this is read, RELOC will prompt the user to insert the BASIC disk. The exec file will then be written.

In order to run the BASIC program now in exec file, it is first necessary to clear any program in the memory witb "NEW." Incidentally, "NEW" (and any other command) may be included in the basic text file, thus making this first step unnecessary. Now load tbe program with the "EXEC" command. The BASIC program is now loaded and ready to be run.

The utility of RELOC extends beyond editing BASIC programs. With slight modification for example, it would be possible to transfer data generated in a FORTRAN or Pascal program to a BASIC program. This would be very handy to anyone desiring Applesoft, Integer BASIC, Pascal, and FORTRAN to have access to some large data base.

MICRO

MAIL TO: IEEE COMPUTER SOCIETY, 10662 LOS VAQUEROS CIRCLE, LOS ALAMITOS, CA 90720.

IEEE Computer Society Membership Application

Check here for your free copy of

☐ *Microprocessors and Microcomputers*

or

☐ *Selected Reprints in Software*

MAILING ADDRESS

First name ______ Middle initial(s) ______ Last name ______

Street address ______

City ______ State/Country ______ Zip ______

Full signature ______ Date ______

OCCUPATION

Title or position ______

Firm name ______

Firm address ______

City ______ State/Country ______ Zip ______

IEEE membership no. or affiliate society membership no. (if applicable) ______ Grade ______

EDUCATION (highest level completed)

Name of educational institution ______

Course ______ Degree received ______ Date ______

ENDORSEMENT (Name one individual, preferably an IEEE member, who knows you professionally.)

Name (print in full) ______

Street address ______

City ______ State/Country ______ Zip ______

FOR OFFICE USE ONLY

Member Number | St/Cntry | Zip Code City | Reg | Coun | Ser

3 10 11 14 20 22 24

Subsec | C/S | Country | A/C | BPA Code | Grade | Birthdate | Sex

26 28 29 32 33 39 40 42 48

Elec Date | Brochure | Year | Disc | Event

72 75

F 4 4 C 8 1 M C R 1 1 - 8 1

1 4 6 7 9 15

schedule of fees

Important: Pay the HALF-YEAR rate if your application is postmarked between March 1, 1982 and August 31, 1982. Pay the FULL-YEAR rate if your application is postmarked between September 1, 1982 and February 28, 1983. Membership and publications expire December 31.

If you are not yet a Computer Society member:	Half-Year	Full-Year
• I am an IEEE member who wishes to join the Computer Society. Enclosed is my	☐ $4.00	☐ $8.00
• I am a non-IEEE member who wishes to join both IEEE and the Computer Society.* Enclosed is my	☐ $44.00	☐ $78.00
• I am a non-IEEE member who wishes to join the Computer Society only. Enclosed is my	☐ $14.00	☐ $28.00
• In addition to my automatic subscription to COMPUTER, I also want to subscribe to *IEEE Micro* (3071). Enclosed is my	☐ $4.00	☐ $8.00
If you are already a Computer Society member:		
• I am already a Computer Society member and wish to subscribe to *IEEE Micro* (3071). Enclosed is my	☐ $4.00	☐ $8.00

Total amount remitted with this application: ______

(Make checks or money orders payable to IEEE. Mail to the IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.)

*IEEE members in Region 7 (Canada) and 8 (Western Europe and the Mid-East) may deduct $6 from full-year rates, $3 from half-year rates. IEEE members in Regions 9 and 10 may deduct $13 from full-year rates, $6.50 from half-year rates. ACM members who join both the IEEE and the Computer Society may deduct $5 from full-year rates, $2.50 from half-year rates. ACM discount applies only to those who join both IEEE and the Computer Society.

Hardcopy bulk air option for members in Regions 8–10: Western Europe, Latin America (including Mexico), and the Far East.

Add the following amount(s) for two-week delivery for COMPUTER and *IEEE Micro*:	Western Europe Half-Year	Western Europe Full-Year	All Other Areas Half-Year	All Other Areas Full-Year
COMPUTER	☐ $3.50	☐ $7.00	☐ $20.00	☐ $40.00
IEEE Micro	☐ $1.75	☐ $3.50	☐ $9.75	☐ $19.50

"Western Europe" includes Albania, Austria, Belgium, Bulgaria, Cyprus, Czechoslovakia, Denmark, Finland, France, East Germany, West Germany, Gibraltar, Greece, Greenland, Holland, Hungary, Iceland, Ireland, Italy, Liechtenstein, Luxembourg, Malta, Monaco, Norway, Poland, Portugal (and Azores), Rumania, Spain (and Canary Islands), Sweden, Switzerland, Turkey, United Kingdom, USSR, and Yugoslavia. "All other areas" includes all other countries outside the U.S., Canada, and Mexico.

Use the numbers to identify your plant activity and your own work. If more than one activity is involved, use the code number for the most prevalent. Note: If you select "other" for work and plant code or "other personnel" for job function code, be sure to explain on the lines provided.

WORK AND PLANT (Enter codes)

Plant ______ Your own work ______

1. Large computers
2. Minicomputers
3. Microcomputers
4. Computer peripheral equipment
5. Data processing systems (system integration)
6. Office and business machines
7. Test, measurement, and instrumentation equipment
8. Communications systems and equip.
9. Navigation and guidance systems and equipment
10. Consumer entertainment electronic equipment
11. Consumer electronic appliances
12. Other consumer electronics
13. Industrial controls, systems, and equip.
14. Components and subassemblies
15. Materials and hardware
16. Aircraft, missiles, space and ground support equipment
17. Oceanography and support equipment
18. Medical electronics
19. Industrial companies within OEM incorporating electronics equip. in their end product, not elsewhere classified
20. Independent research, test, and design laboratory and consultant (only if you are not connected with a mfg. co.)
21. Government agency and military
22. Industrial companies using and/or incorporating any electronic products in their manufacturing, research, or development activities
23. Communication (radio, TV, police)
24. Transportation services (airline, railroads, etc.)
25. Computer and data processing services: service bureaus, software services, timesharing, consulting
26. All other commercial users
27. Power generation equip. manufacturer
28. Power production (atomic, elec., etc.)
29. Power generation
30. Power transmission
31. Power distribution
32. Utilities, except power (telephone, telegraph, pipelines, etc.)
33. Distributor
34. School, university, or library
35. Others (explain)

JOB FUNCTIONS (Enter code) ______

1. General and corporate management
2. Design and development engineering (circuits, components, equip. systems)
3. Engineering services evaluation (quality control, reliability standards, tests)
4. Member of Technical Staff (MTS)
5. Basic research
6. Manufacturing and production
7. Engineering support (draftsman, lab assistant, technician)
8. Purchasing and procurement
9. Marketing, including sales
10. Computer systems/operation
11. Engineering systems planning and design (utilities)
12. Operations including construction and maintenance (utilities)
13. Consulting
14. Other "EE's"
15. Other engineers (explain ______)
16. Dean, professor, instructor, etc.
17. Student
18. Other personnel (explain ______)

JOB CATEGORY (Enter code) ______

1. Management other than engineering
2. Engineering management
3. Engineering

Apple Pascal Textfile Lister

Outputting Language System textfiles to your printer using the transfer command is handy, yet it produces an awkward and untidy listing with no page breaks. This program solves that problem, producing neatly paged output with titles and numbers on each page.

Robert Walker
6100F Wood Chase Lane
Marietta, Georgia 30067

Apple Pascal users are familiar with using the Filer option "Transfer" for printing Language System textfiles. For those desiring program listings with a professional appearance, this method is inadequate. First, each page has no top and bottom margins. In fact, it's common for a line of text to be printed on the perforation between two pages. Second, each page lacks a title and page number, making identification and ordering difficult or impossible if pages become separated. With this in mind, I proceeded to write a short Apple Pascal utility program which included the above mentioned features.

The program employs the following simple algorithm:

Repeat until end of text.
Start at top of page.
Print page heading (page number and optional title).
Scroll past top margin.
Print text until reaching bottom margin.

Although the program is simple and relatively easy to decipher, some explanation is needed.

In the program listing, the constant declaration section contains the most significant items. The constants Topmargin, Bottommargin, and Pagelength represent the number of lines each of these listing parameters occupies. For example, with the values assigned in the listing, 59 lines per page will be available for the textfile (Pagelength-Topmargin-Bottommargin = 66 – 3 – 4 = 59).

The page heading is within the space occupied by the top margin. Specifically, the first line of Topmargin (i.e., the page) will always be blank, followed by the second line which has the page title and number. As a result, the minimum value for Topmargin is two (lines).

For this program to work, your printer must have at least 80 columns, and must be able to vertically tab to the top of a new page when sent a formfeed control character.

To use the program simply eXecute LISTER (or whatever name you prefer). When the textfile name is requested, respond with any valid file name. If you respond with an asterisk, it will be interpreted as "SYSTEM.WRK.TEXT". A return as a response will end the program. Next, the page title will be requested. Once this is entered, the textfile will be listed, producing documentation with a professional appearance.

```
                    (* APPLE PASCAL TEXTFILE LISTER *)
                    (*   Written by Robert Walker   *)
                    (*         Wichita, KS          *)

PROGRAM LISTER;

CONST   Topmargin = 4;  (* minimum of 2 *)
        Bottommargin = 3;
        Pagelength = 66;

VAR     Textfile:  Text;
        Textline, Filename, Pagetitle:  String;
        Linenum, Pagenum, Ioerror:  Integer;
        Printer:  Interactive;


(* Determine if linenum is the last line before the bottom margin. *)
FUNCTION ENDOFPAGE:  Boolean;
Begin
  If Linenum=(Pagelength-Bottommargin)
    then Endofpage:=True
    else Endofpage:=False
End;  (* Endofpage *)


(* Start a new page-- Formfeed, Print heading, and Space up to first line. *)
PROCEDURE STARTNEWPAGE;
```

```
PROCEDURE SPACEUP(Lines:  Integer);   (* send linefeeds to printer *)
VAR I:  Integer;
Begin
  For I:= 1 to Lines do Writeln(Printer)
End;  (* SPACEUP *)
Begin  (* STARTNEWPAGE *)
  If Pagenum<>1 then Page(Printer); (* assume listing begins on new page *)
  Spaceup(1);
  Writeln(Printer,'':(80-Length(Pagetitle)-13) Div 2,Pagetitle,
        '   Page no. ',Pagenum);  (* print page heading *)
  Spaceup(Topmargin-2);
  Linenum:= Topmargin+1;
  Pagenum:= Pagenum+1
End;  (* STARTNEWPAGE *)

Begin (* MAIN PROGRAM *)
  Reset(Printer,'Printer:');
  Writeln(Chr(12),'APPLE PASCAL TEXTFILE LISTER');
  Repeat
    Writeln;
    Write('Textfile Name ("*"=SYSTEM.WRK.TEXT)- ');
    Readln(filename);
    If Filename='' then Exit(LISTER);
    If Filename='*' then Filename:='SYSTEM.WRK.TEXT';
    (*$I-*)
    Reset(Textfile,Filename);
    Ioerror:=Ioresult;
    If Ioerror = 0
      then  (* FILE FOUND *)
        Begin
          Write('Page title- ');
          Readln(Pagetitle);
          Pagenum:= 1;
          Startnewpage;

         (* READ AND WRITE TEXT *)
         While Not Eof(Textfile) do
           Begin
             Readln(Textfile,Textline);
             Writeln(Printer,Textline);
             Linenum:= Linenum+1;  (* Count lines *)
             If Endofpage then Startnewpage
           End;
          Page(printer);  (* formfeed *)
        End
      else  (* I/O ERROR *)
        Begin
          Writeln('I/O ERROR');
          Case Ioerror of  (* MOST COMMON I/O ERRORS *)
             3:  Writeln('ILLEGAL OPERATION');
             7:  Writeln('BAD TITLE');
             9:  Writeln('VOLUME IS NOT ON LINE');
            10:  Writeln('NO SUCH FILE');
            64:  Writeln('BAD ADDRESS OR DATA ON DISK')
          End
        end
  Until Filename=''
End. (* MAIN PROGRAM *)
```

MICRO

MICRObits *(continued)*

Spanish Hangman

2,000 SPANISH words and sentences taught in a fun way on the Apple. Send for your school's free 30-day evaluation diskette, from:

George Earl
1302 South General McMullen
San Antonio, TX 78237

Business Software by ADS

For the Apple II and Atari/800. Why pay more for a bunch of unrelated programs? Business Plus will handle invoices, statements, credit memos and more, much more! Just $299 complete or $25 for demo disk (credited towards purchase). VISA, Mastercharge accepted.

Advanced Data Systems
7468 Maple Avenue
St. Louis, MO 63143
314/781-9388

Extended SYM-BASIC

Adds 30 commands, requires 16K, $85 US/$95 Can., object on cassette, manual, and source listing. SYM-FORTH 1.0: fig-FORTH for 16K SYM-1. Editor, assembler, cassette interface $135 US/$155 Can., object on cassette, manual and source listing.

Saturn Software Limited
8246 116A St.
Delta, B.C., V4C 5Y9
Canada

PET Arcade Software

Astroidz and *Munchman* games for your 8K old-new ROMS. *Astroidz* are invading the galaxy. Four levels of play. *Munchman* is based on arcade game Pac-Man. ZIP and ZAP are out to get you. Fantastic graphics. $9.95 each cassette.

ComputerMat
Box 1664M
Lake Havasu, AZ 86403

PET/CBM Owners

Real world software at low cost. 2114 RAM adapter and 4K Memory Expansion for "old" 8K PETs. Write for free catalog!

Optimized Data Systems
Dept. M, Box 595
Placentia, CA 92670

(Continued on page 107)

Elementary Pascal Internals

Introduction to the internal structure of the P-machine and to some of the concepts which underly the workings of P-code Pascal implementations. While the examples are specific to Abacus Software's Tiny Pascal, others such as UCSD Pascal, Pascal/M, Dynasoft Cassette Pascal, Supersoft Tiny Pascal, and Programma Tiny Pascal are based on similar architectures.

Arnie Lee
Abacus Software
P.O. Box 7211
Grand Rapids, Michigan 49510

Introduction

My interest in the Pascal language stems from two observations and experiences: Pascal is a simple, yet powerful language with which to program, and Pascal is a simple language to implement. These are, in fact, the goals which Wirth set out to meet, as he defined the Pascal language in the late 60's. "The first ... to make available a language suitable to teach programming as a systematic disciple based on certain fundamental concepts clearly and naturally reflected by the language. The second ... to develop implementations of this language which are both reliable and efficient on presently available computers"[1].

Apparently I'm not alone in feeling so positive about Pascal, since micro users have been flocking to use it. If the number of text book titles are any indication of popularity, then we may conclude that Pascal has indeed arrived.

What makes Pascal so attractive? First, the number of language elements in Pascal is not so overwhelming as in other languages such as PL/I. Also, the elements are not cryptic as with the APL language. Yet even with a relatively modest vocabulary, Pascal affords a concise and clear language with which to program solutions to problems. Thus, the demand for Pascal is due, in part, to its simplicity and usefulness for a variety of applications.

This great demand for Pascal is also easily satisfied, as the language is relatively simple to implement. Pascal has been implemented on most popular micros, and some versions will run in as little as 16K of memory. I have implemented a subset of the full Pascal language for the PET/CBM and Apple II. This undertaking required only modest effort. In the remainder of this article, I will share some of what I learned about Pascal internals.

Pascal Compilers and Interpreters

The statements of a high-level language such as Pascal are called a source program. The Pascal compiler translates the source program into a semantically equivalent program that can later execute on the micro.

Some Pascal compilers translate the source program into machine code, which is directly executable by the computer. Most micro-based Pascal compilers translate the source program into P-code. P-code, however, cannot be directly executed by the computer. Instead, it needs to be interpreted.

Interpreters spend a large amount of execution time determining the desired operations to be carried out on behalf of the program. (Most implementations of BASIC are interpreters.) A compiler, on the other hand, determines the operations to be performed and creates code that reflects these operations. Then the code is executed at a later time. Since the operations are "predetermined" beforehand, a compiled program generally executes faster than an interpreted program. Pascal systems such as UCSD Pascal, Pascal/M and Tiny Pascal, combine both concepts of compilation and interpretation. So, the source is compiled, which predetermines the operations to be performed, but the resultant P-code is interpreted by the P-machine.

The P-Machine

Most micro implementations of Pascal are based on a 16-bit pseudo-machine model. The pseudo-machine, often called the P-machine, "executes" P-code as its machine language. The P-machine performs its computations through stack operation.

A stack is a data structure that is used to store data. The stack consists of multiple "slots" into which data may be stored, and from which the same data may be retrieved. The *stack pointer* always points to the next slot available for data storage.

There are two operations that may be performed on the stack. PUSHing a value onto the stack means that a given data value is stored into the slot indicated by the stack pointer. After the value is PUSHed, the stack pointer is altered to point to the next available slot. Conversely, POPing a given data value means that the stack pointer is altered to point to the last used slot, and the value occupied by that slot is returned to the program.

The P-machine is actually an *interpreter* that is written in the assembly language of the micro. This interpreter simulates the operations of the hypothetical stack-oriented computer. The interpreter handles all of its operations as if it were a 16-bit computer by emulating all of its elementary operations (arithmetic, comparisons, input, etc.) using 16-bit data and stack operations. Each slot of the stack can accommodate a 16-bit data item. When executing P-code "instructions," the P-machine expects the operands to be on the stack. The P-machine then carries out the computations and places the results back onto the stack.

Figure 1 is an example of a typical stack operation. We want to add two variahles (B and C) and assign the sum to a third variable (A). The instruction is shown on the left, wbile the contents of the stack after the operation is shown on the right.

Instruction 1 shows the stack before any operations take place. *Instruction 2* PUSHes the value of variable B onto the stack, therehy incrementing the stack pointer. *Instruction 3* PUSHes the value of variable C onto the stack, also incrementing the stack pointer. *Instruction 4* POPs the value of C off the stack, POPs the value of B off the stack, performs the addition, and finally PUSHes the sum onto the stack. The stack pointer is altered to point to the next available slot (previously occupied by C). Finally, *instruction 5* POPs the sum off the stack and stores the result in variable A. The stack pointer is left pointing to the same slot as hefore the series of operations was executed.

Figure 1: Typical Stack Operation

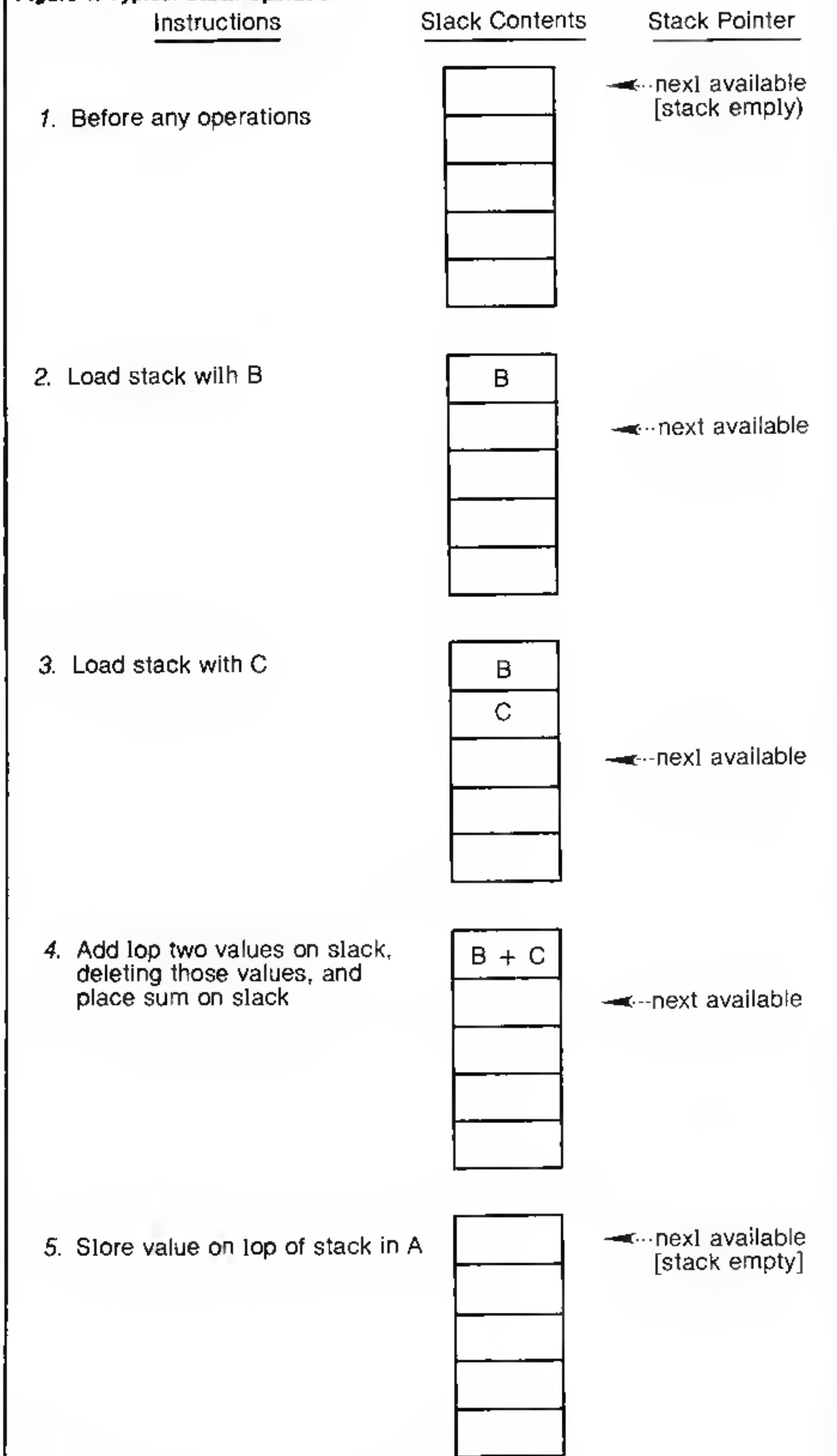


Figure 2

	Resultant P-codc:	
A := B + C;	LOD B	load stack with value of B
	LOD C	load stack with value of C
	OPR ADD	add two values on top of stack
	STO A	store top of stack in A

P-Code Instructions

Instructions direct the P-machine to perform certain stack operations. These instructions are referred to as "P-code instructions" or just "P-code." It is the job of the *compiler* to convert the Pascal source statements to "P-code." After analyzing the Pascal source statements, the compiler generates the appropriate P-code to perform the desired program operations. This P-code is later executed by the P-machine. Execution in this case really means that the P-code is interpreted hy the P-machine.

To continue the above example, let's look at the P-code produced by the Pascal language source statement in figure 2.

This is the example shown previously, using the mnemonics of a typical P-machine. Table 2 describes the mnemonics that are used in some typical Pascal implementations. Below is another example for a more complex arithmetic statement:

A := B + C DIV (D - 6);

In figure 3, the compiler bas to work a little harder to analyze the statement and generate the appropriate P-code. The operator precedence directs the compiler to first generate the P-code for the DIVide operation. (Division is performed before addition hecause division has a higher operator precedence. See table 1.) Because of the parenthetical expression (D - 6), the P-code for that operation must be generated before the

Figure 3

```
LOD  B ──────────────────────────────┐
LOD  C ────────────────┐             │
LOD  D ──┐             │             │
LIT  6   ├→ D-6 ├→ C DIV (D - 6)     ├→ B + C DIV (D - 6)
OPR  SUB─┘             │             │
OPR  DIV───────────────┘             │
OPR  ADD─────────────────────────────┘
STO  A
```

***Table 1:* Operator Precedence**

NOT	highest
* DIV MOD AND	
+ − OR	
= <> < <= > >=	lowest

***Table 2:* P-Code Instructions**

The following table is a list of the P-code instructions that are used by this version of TINY Pascal PLUS +. The P-codes represent the instruction set of a hypothetical 16-bit stack machine similar to the one described by Wirth in his ALGORITHMS + DATA STRUCTURES = PROGRAMS.

Opcode (Hex)	Mnemonic	Operand (Dec)	Description
00	LIT	n	Load literal constant to (SP)
10	OPR	0	Return from procedure
10	OPR	1	Negate (SP)
10	OPR	2	Add (SP) to (SP − 1)
10	OPR	3	Subtract (SP) from (SP − 1)
10	OPR	4	Multiply (SP − 1) by (SP)
10	OPR	5	Divide (SP − 1) by (SP)
10	OPR	6	Test for odd
10	OPR	7	(SP − 1) modulo (SP)
10	OPR	8	Test for (SP − 1) = (SP)
10	OPR	9	Test for (SP − 1) > (SP)
10	OPR	10	Test for (SP − 1) < (SP)
10	OPR	11	Test for (SP − 1) > = (SP)
10	OPR	12	Test for (SP − 1) < (SP)
10	OPR	13	Test for (SP − 1) < = (SP)
10	OPR	14	(SP − 1) OR (SP)
10	OPR	15	(SP − 1) AND (SP)
10	OPR	16	NOT (SP)
10	OPR	17	Shift left (SP)
10	OPR	18	Shift right (SP)
10	OPR	19	(SP) + 1
10	OPR	20	(SP) − 1
10	OPR	21	Copy (SP) to (SP + 1)
20 + v	LOD	d	Load (SP)
20 + X'F'	LOD	0	Load from absolute addr (SP)
30 + v	STO	d	Store (SP)
30 + X'F'	STO	0	Store at absolute addr (SP)
40 + v	CAL	a	Procedure call
40 + X'F'	CAL	0	Call proc at absolute addr (SP)
50	INT	n	Increment stack pointer by n
60	JPC	a	Jump to location a if low order bit of (SP) = 0
61	JPC	a	Jump to location a if low order bit of (SP) = 1
6F	JPC	a	Jump unconditionally to location a
70	CSP	0	Input a single character
70	CSP	1	Output a single character
70	CSP	2	Input an integer
70	CSP	3	Output an integer
70	CSP	4	Input a hexadecimal number
70	CSP	5	Output a hexadecimal number

(Continued)

P-code for the DIVide operation is completed. Finally, because of operator precedence rules, the P-code for the ADDition operation is generated.

More Examples

Listing 1 is a sample Pascal source program that performs some elementary arithmetic operation. The program does not perform any useful work and is only meant to illustrate the following examples.

Listing 2 shows the P-code that is generated by the compiler for the Pascal program of listing 1. The listing shows the Pascal source statements and the respective P-code instructions generated for that statement. The listing also shows the memory location at which the P-code is temporarily stored and the internal format of the P-code.

Listing 3 is a Pascal program called DISASSEM which can disassemble the P-code. This program disassembles only the P-code generated by the TINY Pascal PLUS + compiler, but the same technique may be used to disassemble any P-code, provided that the user is aware of the internal format of his Pascal system's P-codes. Table 2 shows the P-code formats for this implementation.

Listing 4 is a sample disassembly of the Pascal program from listing 1. Listing 4 was produced by the DISASSEM program.

Arnie Lee is a data base analyst for a large manufacturing firm in the Grand Rapids, Michigan area. He is interested in all types of computer languages for micros, and hopes to develop data base systems for these smaller machines. He is co-author of *Tiny Pascal* and the author of the *PET Machine Language Guide*.

Table 2 *(Continued)*

70	CSP	6	Read keyboard without return key
70	CSP	8	Output a string
70	CSP	9	GRAPHICS set graphics mode for or Lo-Res graphics mode for Apple II
70	CSP	10	COLOR sets color for Apple II or PET
70	CSP	11	PLOT point at (SP – 1,SP) for PET and Apple II Lo-Res graphics
70	CSP	12	POINT function at (SP – 1,SP). For PET returns 0 if point is off and returns 1 if point is on. For Apple II returns the number of the Lo-Res color.
70	CSP	13	TEXT sets text (non-graphics) mode
70	CSP	14	HGRAPHICS sets Apple II Hi-Res graphics mode
70	CSP	15	HCOLOR sets color for Apple II Hi-Res graphics
70	CSP	16	HPLOT Hi-Res plot at (SP – 1,SP) for Apple II
70	CSP	17	HPLOT Hi-Res plot from (SP – 3, SP – 2) to (SP – 1,SP) for Apple II
70	CSP	18	ABS(SP) absolute value function
70	CSP	19	SQR(SP) square of value
70	CSP	20	PDL(SP) read paddle
70	CSP	21	TONE play tone on speaker
A0 + v	LODX	d	Load with index (SP)
B0 + v	STOX	d	Store with index (SP)

The stack pointer is SP. It points to the top element on the stack. (SP) represents the contents of the top element. The address a is a P-code address. The displacement d is the displacement from a base address. The static level difference v is used by procedure calls to isolate variables. The number n is a numeric constant.

Listing 1

```
(*  SAMPLE PASCAL PROGRAM TO DEMONSTRATE SOME
    OF THE ARITHMETIC STATEMENTS IN OF THIS
    IMPLEMENTATION OF PASCAL.              *)

CONST CR=13;

VAR   A,B,C: INTEGER;

BEGIN
   A := 0;          (* CONSTANT ASSIGNMENT *)
   A := B;          (* VARIABLE ASSIGNMENT *)
   A := B + C;      (* ADDITION *)
   A := B - C;      (* SUBTRACTION *)
   A := B * C;      (* MULTIPLICATION *)
   A := B DIV C;    (* DIVISION *)
   A := B MOD C;    (* MODULO DIVISION *)
   A := SQR(B);     (* SQUARE FUNCTION *)
END.

*** 18 LINES IN FILE ***
```

Listing 2

```
       TINY PASCAL PLUS+       FROM ABACUS SOFTWARE


ADDR   REL.  CODE     OPND. ADDR   SOURCE CODE

                                0   (*  SAMPLE PASCAL PROGRAM TO DEMONSTRATE SOME
                                0       OF THE ARITHMETIC STATEMENTS IN OF THIS
                                0       IMPLEMENTATION OF PASCAL.              *)
                                0
                                0   CONST CR=13;
8600    0    JPC       15     0
```

(Continued)

MICRObits *(continued)*

The Index

Over 12,000 magazine and newsletter articles, editorials, and columns from magazines such as MICRO, *BYTE*, *COMPUTE!*, *Interface Age*. Over 40 magazines and newsletters indexed by machine types like Apple, Atari, TRS-80, and more. Order from your local computer store. Price $14.95 plus 50¢ shipping.

Missouri Indexing
P.O. Box 301
St. Ann, MO 63074
(213) 997-6470

OSI Software

Universe (by the author of *Galaxian*) is a super smooth arcade game played by a keyboard or joystick — $14.95. *Edit All* includes a full screen editor, scroll window output, full cursor control and more — $19.95. For C1P, C2-4 systems.

DMP Systems
319 Hampton Boulevard
Rochester, NY 14612

Ohio Scientific C4P

Games: *Galactic Trader, Drag Racer II, Missile Defense, Shuffle Bowling, Lunar Lander* and more! Many with graphics, real time action, color and sound. Other programs: *Transformer Design, Home Budget*. Send SASE for catalog to:

Ron Lashley Software
2934 W. Missionwood Circle
Miramar, FL 33025

OSI Non-Symbolic Disassembler

Now look at machine code programs in disassembled format. Standard 6502 mnemonics. Screen control features, step through memory to look at ROM and RAM routines. Runs in under 2K memory. Autoload cassette only $9.95. A *real* programmer's tool.

Computer Science Engineering
10 Liberty Avenue
Burlington, MA 01803

Medical Computer Journal

The journal for computer use in daily practice. Programs, system description, laboratory, original articles. Bound Vol. 1 — $17.00, subscriptions — $15.00/year.

Medical Computer Journal
A. Ghaussey
42 E. High Street
East Hampton, CT 06424

(Continued on page 109)

New MICRO Department

- If you have an idea which is not suited for a lengfhy article, but is nonetheless interesting and useful, submit it to MICRO's Short Subjects department.
- Wrife-ups for this section should be brief and concise. Cover your topic in just a few paragraphs. We'll also accept short listings.
- Contributors to this section will not be paid, but will receive full credit and acknowledgement with a byline.

Listing 2 *(Continued)*

```
                                    1
                                    1 VAR   A,B,C: INTEGER;
                                    1
                                    1 BEGIN
                                    1   A := 0;        (* CONSTANT ASSIGNMENT *)
                                          ****ADDR AT 0 CHANGED TO 1
8603   1    INT    0   6
8606   2    LIT    0   0
8609   3    STO    0   3
                                    4   A := B;        (* VARIABLE ASSIGNMENT *)
860C   4    LOD    0   4
860F   5    STO    0   3
                                    6   A := B + C;    (* ADDITION *)
8612   6    LOD    0   4
8615   7    LOD    0   5
8618   8    OPR    0   2
861B   9    STO    0   3
                                    10  A := B - C;    (* SUBTRACTION *)
861E   10   LOD    0   4
8621   11   LOD    0   5
8624   12   OPR    0   3
8627   13   STO    0   3
                                    14  A := B * C;    (* MULTIPLICATION *)
862A   14   LOD    0   4
862D   15   LOD    0   5
8630   16   OPR    0   4
8633   17   STO    0   3
                                    18  A := B DIV C;  (* DIVISION *)
8636   18   LOD    0   4
8639   19   LOD    0   5
863C   20   OPR    0   5
863F   21   STO    0   3
                                    22  A := B MOD C;  (* MODULO DIVISION *)
8642   22   LOD    0   4
8645   23   LOD    0   5
8648   24   OPR    0   7
864B   25   STO    0   3
                                    26  A := SQR(B);   (* SQUARE FUNCTION *)
864E   26   LOD    0   4
8651   27   CSP    0   19
8654   28   STO    0   3
                                    29    END.
8657   29   OPR    0   0

***   LENGTH OF P-CODE IS 92   ***
```

Listing 3

```
(*     DISASSEM - A P-CODE DISASSEMBLER

       BY ARNIE LEE
       ABACUS SOFTWARE
       P.O.  BOX  7211
       GRAND RAPIDS, MI 49510

       THIS PROGRAM WILL WORK ON EITHER THE PET OR APPLE II.
       IT ASSUMES THAT THE P-CODE FILE TO BE DISASSEMBLED
        HAS ALREADY BEEN LOADED BY AN ALTERNATIVE METHOD.

                                         *)
CONST CR=13;
       FALSE=1;
       TRUE=0;

VAR    STARTLOC,NUMINSTR,
       LASTPCODE,PCTR,INSTR,
       MODIFIER,OPERAND:INTEGER;

BEGIN
  NUMINSTR:=0;
  LASTPCODE:=FALSE;
  WRITE('ENTER P-CODE STARTING LOCATION-> ');
  READ(PCTR%);
  WHILE LASTPCODE=FALSE DO
    BEGIN
      INSTR:=MEM(PCTR) SHR 4;
      MODIFIER:=((MEM(PCTR)) SHL 12) SHR 12;
      OPERAND:=MEM(PCTR+1) + MEM(PCTR+2) SHL 8;
      NUMINSTR:=NUMINSTR + 1;
      WRITE(PCTR%,' ',MEM(PCTR)%,' ',MEM(PCTR+1)%,' ',MEM(PCTR+2)%,' ');
```

MICRObits *(continued)*

OSI BASIC LOAD & SAVE w/Filename

Tired of LOAD SAVE times, tape documentation? Cry no more. Now SAVE with filenames on your cassette programs. LOAD by filename. Gives directory of tape as you go. Employs a token I/O system to decrease tape I/O by 50% — $10.95.

Computer Science Engineering
57 Beals Street, Rm. 57-12
Brookline, MA 02146

Graphics Utility for the Apple II

Ampergraph adds 21 Applesoft commands which allow effortless generation of graphs of scientific or financial data. The imbedded commands include &SCALE, &LIMIT, &AXES, &GRID,&FRAME, &LOGX, &LOGY, &LABELAXES, &LABEL, &DRAW, &PENUP, &ERRORBARS, etc. System: DOS 3.3, 48K — $25.00 ($1.00 shipping).

Madwest Software
Department M
P.O. Box 9822
Madison, WI 53715

Commodore VIC 20

3K Memory Expansion — $59.95 ppd. Plugs in in seconds. VIC Graphic Manual — $11.95. Most complete documentation on Hi-Res with tape. Joystick Software: *Spider's Parlor, Packman, U.F.O., Asteroids, SJ Slither* — $5.95 each. SASE brings information.

Peripheral Products
3030 So. Vermont Avenue
Royal Oak, MI 48067

Wanted: OSI 300 board and...

early 400 series boards, documentation and catalogs from 1977 and earlier.

Edward H. Carlson
3872 Raleigh Drive
Okemos, MI 48864
(517) 349-1219

Adventure #1 — Murder Mansion

You and six other people are exploring a three-story mansion for a hidden treasure. But *watch out*! Someone has decided to get *greedy* and is killing the others. You could be next! 8K OSI C1P-C4P — $12.95. 32K TRS-80 Model I and III (graphics) — $15.95.

Comp-U-Game Software
P.O. Box 802
Nevada, MO 64772

MICRO

Listing 3 *(Continued)*

```
       CASE INSTR OF
         0:  WRITE('LIT ',MODIFIER#,' ',OPERAND#,CR);
         1:  BEGIN
               WRITE('OPR ');
               CASE OPERAND OF
                  0:  WRITE('RETURN');
                  1:  WRITE('NEGATE');
                  2:  WRITE('ADD');
                  3:  WRITE('SUBTRACT');
                  4:  WRITE('MULTIPLY');
                  5:  WRITE('DIVIDE');
                  6:  WRITE('TEST ODD');
                  7:  WRITE('MODULO');
                  8:  WRITE(' = ');
                  9:  WRITE(' <> ');
                 10:  WRITE(' < ');
                 11:  WRITE(' >= ');
                 12:  WRITE(' > ');
                 13:  WRITE(' <= ');
                 14:  WRITE(' OR ');
                 15:  WRITE(' AND ');
                 16:  WRITE(' NOT ');
                 17:  WRITE(' SHL ');
                 18:  WRITE(' SHR ');
                 19:  WRITE(' +1 ');
                 20:  WRITE(' -1 ');
                 21:  WRITE('COPY')
                 ELSE WRITE('INVALID')
               END;
               WRITE(CR);
             END;
         2:  WRITE('LOD ',MODIFIER#,' ',OPERAND#,CR);
         3:  WRITE('STO ',MODIFIER#,' ',OPERAND#,CR);
         4:  WRITE('CAL ',MODIFIER#,' ',OPERAND#,CR);
         5:  WRITE('INT ',MODIFIER#,' ',OPERAND#,CR);
         6:  WRITE('JPC ',MODIFIER#,' ',OPERAND#,CR);
         7:  BEGIN
               WRITE('CSP ');
               CASE OPERAND OF
                  0: WRITE('INCHR');
                  1: WRITE('OUTCHR');
                  2: WRITE('INNUM');
                  3: WRITE('OUTNUM');
                  4: WRITE('INHEX');
                  5: WRITE('OUTHEX');
                  6: WRITE('INKEY');
                  8: WRITE('OUTSTR');
                  9: WRITE('GRAPHICS');
                 10: WRITE('COLOR');
                 11: WRITE('PLOT');
                 12: WRITE('POINT');
                 13: WRITE('TEXT');
                 14: WRITE('HGRAPHICS');
                 15: WRITE('HCOLOR');
                 16: WRITE('HPLOT X,Y');
                 17: WRITE('HPLOT X,Y TO A,B');
                 18: WRITE('ABS');
                 19: WRITE('SQR');
                 20: WRITE('PDL');
                 21: WRITE('TONE')
                 ELSE WRITE('INVALID')
               END;
               WRITE(CR);
             END;
      10:  WRITE('LODX',MODIFIER#,' ',OPERAND#,CR);
      11:  WRITE('STOX',MODIFIER#,' ',OPERAND#,CR);
      15: BEGIN LASTPCODE:=TRUE; WRITE('EOF ',CR) END
       ELSE WRITE('INVALID P-CODE INSTR.')
    END;
  PCTR:=PCTR + 3;
  NUMINSTR:=NUMINSTR + 1;
  END;
  WRITE('# INSTRUCTIONS DECODED-->',NUMINSTR#,CR);
END.

*** 106 LINES IN FILE ***
```

Listing 4

```
           ABACUS SOFTWARE
        TINY PASCAL INTERPRETER

ENTER FILE NAME OF P-CODE DISASSEM

TRACE ACTIVE(Y/N)?N

ENTER P-CODE STARTING LOCATION-> 8D00

 8000 006F 0001 0000 JPC 15 1
 8003 0050 0006 0000 INT 0 6
 8006 0000 0000 0000 LIT 0 0
 8009 0030 0003 0000 STO 0 3
 800C 0020 0004 0000 LOD 0 4
 800F 0030 0003 0000 STO 0 3
 8012 0020 0004 0000 LOD 0 4
 8015 0020 0005 0000 LOD 0 5
 8018 0010 0002 0000 OPR ADD
 801B 0030 0003 0000 STO 0 3
 801E 0020 0004 0000 LOD 0 4
 8021 0020 0005 0000 LOD 0 5
 8024 0010 0003 0000 OPR SUBTRACT
 8027 0030 0003 0000 STO 0 3
 802A 0020 0004 0000 LOD 0 4
 802D 0020 0005 0000 LOD 0 5
 8030 0010 0004 0000 OPR MULTIPLY
 8033 0030 0003 0000 STO 0 3
 8036 0020 0004 0000 LOD 0 4
 8039 0020 0005 0000 LOD 0 5
 803C 0010 0005 0000 OPR DIVIDE
 803F 0030 0003 0000 STO 0 3
 8042 0020 0004 0000 LOD 0 4
 8045 0020 0005 0000 LOD 0 5
 8048 0010 0007 0000 OPR MODULO
 804B 0030 0003 0000 STO 0 3
 804E 0020 0004 0000 LOD 0 4
 8051 0070 0013 0000 CSP SQR
 8054 0030 0003 0000 STO 0 3
 8057 0010 0000 0000 OPR RETURN
 805A 00FF 00FF 0000 EOF
 # INSTRUCTIONS DECODED-->62

*** EXIT PASCAL INTERPRETER ***
3255 INSTRUCTIONS EXECUTED
HIGHEST STACK ADDRESS USED = $
```

MICRO

MICRO

Dr. William R. Dial
438 Roslyn Avenue
Akron, Ohio 44320

6502 Bibliography: Part XXXX

1153. Compute! 3, No. 4, Issue 11 (April, 1981)

Stewart, C.A., "OSI C1P Control Functions," pg. 110-113.
A method to implement commands (RUN, LOAD, SAVE, LIST, etc.) by a single keystroke.

Mason, James L., "Double the Cassette Baud Rate of Your OSI Superboard II/1P," pg. 114-115.
A hardware modification for the Superboard.

Butterfield, Jim, "Relative File Mechanics," pg. 122-124.
An instructional article on relative files for the PET.

Hinrichs, William L., "Matrix Row Operations," pg. 56-63.
A mathematics classroom teaching aid (Apple/PET) including a program in BASIC.

DeJong, Marvin L., "A Floating Point Binary to BCD Routine," pg. 66-73.
Machine language routines for 6502 micros.

Harvey, Francis A. and Collins, Rosann W. and Hines, Theodore C., "Fill the Screen with Your Message," pg. 80-86.
Generating large multi-colored characters using Apple low-resolution graphics.

Kelly, Derek, "Decrementing the FOR...NEXT and Endless Loops," pg. 88-90.
An instructional article for Applesoft users.

Stewart, Ed., "Unleash the Power of Your Atari CPU," pg. 102.
How to get a 30% increase in speed from your Atari CPU.

Brannon, Charles, "String Arrays in Atari BASIC," pg. 103-104.
A method to simulate string array handling in Atari BASIC.

Butterfield, Jim, "Working with BASIC 4.0," pg. 136.
Tips on getting printed output from the machine language monitor in PET's BASIC 4.0.

Butterfield, Jim, "Disabling the STOP Key," pg. 141.
How to disable the PET's STOP key.

Turco, Francis, "Dissecting C.W. Moser's ASSM/TED 1.0," pg. 142-144.
Assistance for the user of this PET utility.

Deal, Elizabeth, "How to Get Started in Machine Code," pg. 146-151.
For PET users taking their first steps in machine code.

Herman, Harvey B., "KIM Tidbits," pg. 163-164.
Expanding the KIM system.

Olivo, Richard F., "Printing a Symbol Table for the AIM 65 Assembler," pg. 165.
Assistance for AIM 65 users.

1154. Interface Age 6, Issue 5 (May, 1981)

Baker Al, "Game Corner," pg. 27-29.
An Apple program for measuring the readability (Fog Index) of plain text.

1155. Peelings II 2, No. 2 (March/April, 1981)

Staff, "Apple Software Evaluation," pg. 5-44.
Reviews of 11 educational programs, 4 communications packages, 3 utilities and several miscellaneous programs.

1156. OSI User's Independent Newsletter, No. 9 (April, 1981)

Anon., "Printer Driver," pg. 6.
Source code for a driver for a serial EMAKO printer and CA-10X card.

Anon., "PEEK and POKE," pg. 7-8.
PEEK and POKE for Pascal on the OSI systems.

1157. Nibble 2, No. 2 (April, 1981)

Thompson, C.J., "Applesoft NIFFUM," pg. 49-53.
NIFFUM.FP for the Apple will transfer Applesoft files from DOS 3.3 to 3.2 diskettes.

Mottola, R.M., "Inputting Strings with Commas," pg. 57.
A utility for the Apple.

Black, Preston R., "Disk Snooping — Part 1," pg. 63-70.
The first of a series on disk utilities for the Apple disk.

Thompson, C.J., 'BSAVE Can be Easy," pg. 71.
How to figure out the A$ and L$ parameters for binary saves on the Apple.

Reynolds, William III, "Using Text Files from Machine Language, pg. 71-74.
Tips on using Apple text files.

Chipchase, Frank D., "Chaining Applesoft Without a Chain," pg. 77.
A technique for Apple programmers.

1158. Recreational Computing 9, No. 6, Issue 51 (May/June, 1981)

Thornburg, David D., "Pilot's Turtle Graphics for Atari," pg. 16-20.
Atari Pilot offers a simple way to develop graphics on the Atari.

1159. Spreadsheet 1, Issue 3 (March, 1981)

Barker, Thomas, "Visitip #7," pg. 1.
How to set up a label to your worksheet on the prompt line, on VisiCalc.

Bostater, John, "Introducing a Bit of Logic into a Model," pg. 5.
Here is an algorithm for VisiCalc to match numbers, with an example of sorting out information by a code number.

1160. POKE-Apple 3, No. 3 (April, 1981)

Holderby, Michael and Hamilton, Al, "Integer BASIC Token Scheme," pg. 10-14.
An instructional article on Apple's use of tokens.

1161. Mini'App'Les 4, No. 4 (April, 1981)

Anon., "Driver Demo," pg. 9.
A BASIC driver demo program to send block graphics to an Epson MX-80 with modified Apple Epson board.

Anon., "Dan On Printers," pg. 13.
Some tips on using the Epson MX-80.

Murrell, Mike, "Addendum to Dan on Printers," pg. 13.
Notes on using the SSS AIO board with the MX-80.

Thiesfeld, Chuck, "Chuck on Printers," pg. 14-16.
How to modify the Epson Apple interface board so your MX-80 can print block graphics.

1162. Apple-Can 1, No. 1 (December, 1980)

Radford, Bob, "Pascal with One Drive," pg. 15-16.
How to improve the one-drive system by breaking up the Apple/Pascal system programs among a number of disks according to the use that is made of each disk.

1163. Apple-Can 1, No. 2 (February, 1981)

Falkner, Keith, "The Truth of Magnetism," pg. 17-18.
An interesting article dispelling some of the myths about tapes or disks being excessively sensitive to magnetic fields. It takes a very substantial field strength to cause loss of data!

Pugh, John, "Soft Copy to Screen," pg. 24.
A utility for users of the Magic Window word processing system.

Hurd, John C., "The Sorcerer: A Program for Converting Machine Code to Source Text," pg. 28-32.
A utility for the user of Apple assembly language.

1164. Apple-Can 1, No. 3 (April, 1981)

Wojdylo, Tomasz, "Apple Repair Tips," pg. 14.
Some tips on what to look for when your disk drive gives up and issues a burnt plastic odor!

Vella-Zarb, Pierre, "Sorcerer_Extension: A Cheap Disassembler," pg. 23-28.
A utility for the Apple.

1165. Atari Computer Enthusiasts 2, Issue 4 (April, 1981)

Goff, Stacy A., "Fileindx — Part Two," pg. 10-11.
A utility to allow you to index your disks and directories, for the Atari.

1166. Apple Assembly Line 1, Issue 7 (April, 1981)

Sander-Cederlof, Bob, "Text File in Assembly Language Programs," pg. 2-4.
How to read or write a text file from your Apple assembly language program.

Sander-Cederlof, Bob, "Applesoft Entry Points," pg. 4-5.
A useful table for Apple programmers.

Sander-Cederlof, Bob, "Commented Listings of DOS 3.2.1 and #.# Formats," pg. 11-17.
The Format routines of the two DOS for the Apple disks are disassembled and explained.

1167. The Apple-Dillo (April, 1981)

Sethre, Tom, "Program Utility Printer."
This program assists you in printing listings of programs in a wider format than appears on your screen — for the Apple.

1168. Electronics 54, No. 6 (March 24, 1981)

Wyatt, Michael A., "Laplace Transform for tbe Apple," pg. 163.
A time domain display of a Laplace transform.

1169. The G.R.A.P.E. Vine 2, No. 3 (April, 1981)

Lawson, Steve, "13- and 16-Sector DOS," pg. 5.
How to cope with 13- and 16-sector disks using several approaches.

1170. Dr. Dobb's Journal 6, No. 55, Issue 5 (May, 1981)

Pittman, Tom, "Analysis of an Analysis," pg. 4.
Discussion of 6502 opcodes.

1171. The Apple Barrel 4, No. 2 (April, 1981)

Gilbreth, Lee, "Son of a ... File Cabinet," pg. 5-6.
A round-up of modifications for the popular Apple utility, File Cabinet.

1172. PEEK(65) 2, No. 4 (April, 1981)

Williams, Jim, "Two Dollar BSR X-10 Interface," pg. 5-6.
Hardware and software for a wireless AC remote control system.

Loos, James, "C1P as a Dumb Terminal," pg. 6-8.
Machine language routine for OSI systems such as C1P.

Jones, David A., "Cassette Corner," pg. 8, 21-22.
Notes on saving machine language code on OSI systems.

1173. F.W.A.U.G. Newsletter 2, No. 4 (February, 1981)

Meador, Lee, "Disassembly of DOS 3.2," pg. 2-6.
A continuation of this popular series.

1174. T.A.R.T. Newsletter 2, Issue 2 (April, 1981)

Sand, Paul A. and Koerin, Sid, "Pretty Listing," pg. 22-29.
A program to modify your program listings to reformat into a more readable form.

1175. The Apple Orchard 2, No. 1 (Spring, 1981)

Weiglin, Peter C., "Screen Formatting of Text," pg. 9-14.
An instructional article for Apple owners.

Good, R.H., "Practical Super Hi-Res Graphics," pg. 20.
Plot on the Apple with a resolution of 560 × 192.

Reynolds, Lee, "S.H. Lam Routine Utility," pg. 28-29.
A routine to set up machine language routines within a BASIC program on the Apple.

Nareff, Max J., "Matrix Functions with the Apple," pg. 36-38.
How to reconstruct those matrix functions missing from Applesoft.

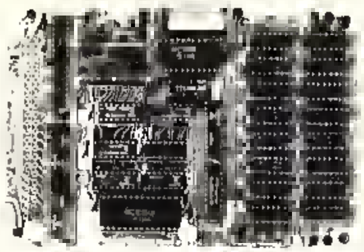
Mack, Art, "Sorting in BASIC," pg. 68.
A simple sort routine for Apple.

Golding, Val J., "Proper Printer Protocol," pg. 91.
A suggested standard BASIC printer call subroutine.

1176. Abacus II 3, Issue 3 (March, 1981)

Howard, Clifton M., "Directory Title Formatting," pg. 5-24.
A concept of directory title formatting including a series of utilities for disk management for the Apple.

MICRO



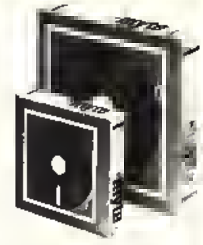
BETA 32K BYTE EXPANDABLE RAM FOR 6502 AND 6800 SYSTEMS
AIM 65 KIM SYM PET S44-BUS

- Plug compatible with the AIM-65/SYM expansion connector by using a right angle connector (supplied).
- Memory board edge connector plugs into the 6800 S44 bus.
- Connects to PET using an adaptor cable.
- Uses +5V only, supplied from the host computer.
- Full documentation. Assembled and tested boards are guaranteed for one full year. Purchase price is fully refundable if board is returned undamaged within 14 days.

Assembled with 32K RAM	$349.00
& Tested with 16K RAM	329.00
Bare board, manual & hard-to-get parts	99.00
PET interface kit. Connects the 32K RAM board to a 4K or 8K PET	$ 69.00

See our full-page ad in BYTE and INTERFACE AGE

wabash



8" or 5¼" flexible diskettes certified 100% error free with manufacturer's 5-year limited warranty on all 8" media. Soft-sectored in boxes of 10. 5¼" available in 10-sector.

(Add $3.00 for plastic library cases)

8" single sided, single density	$27.50
8" single sided, double density	35.50
8" double sided, double density	45.50
5¼" single sided, single density	27.50
5¼" single sided, double density	29.50
5¼" single sided, double density, 10-sector	$29.50

TERMS: Minimum order $15.00. Minimum shipping and handling $3.00. Calif. residents add 6% sales tax. Cash, checks, Mastercard, Visa and purchase orders from qualified firms are accepted. (Please allow two weeks for personal checks to clear before shipment.) Product availability and pricing subject to change without notice.

INTERNATIONAL ORDERS: *Add 15% to purchase price for all orders. Minimum shipping charge is $20.00. Orders with insufficient funds will be delayed. Excess funds will be returned with your order. All prices are U.S. only.*

PAPER TIGER PRINTERS

IDS 460G 9x9 Dot Matrix Printer	$890.00
IDS 560G Wide Carriage Printer	1099.00

TERMINALS

ADDS Viewpoint	$569.00
TeleVideo 910	579.00
TeleVideo 912C	679.00
TeleVideo 920C	729.00
TeleVideo 950	929.00

8" DISK DRIVES

Shugart 801R	$399.00
NEC FD1160 (double sided)	569.00

DYNAMIC RAMS

4116 (200ns)	set of 8	$24.00
4164 (64Kx1)		$18.00

BETA COMPUTER DEVICES
1230 W.COLLINS AVE.
ORANGE, CA 92668
(714) 633-7280

---

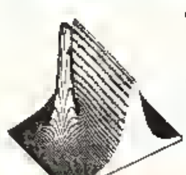
560 DOTS/LINE

Double Density HGR

COSMOS SCREEN MIXER

NORMAL
HIGHLIGHT

Highlight



THIS IS TEXT

Hi-Res Graphics+Text

THIS IS TEXT
LGR

Lo-Res Graphics+Text

SCREEN MIXER is a set of three modules for APPLE-II. Just plug-in these modules to your Apple-II, and you will have the Apple-II with more features you could not expect till now----

SCREEN MIXER provides:

■The mixed screen of any two of screens available for the Apple-II. Please note that all of HGR, LGR and Text screen has two pages. The mixing is done with hardware, not like Hi-Res Text Generater Programs, thus you need no software and the scroll speed is not reduced. Also, you can scroll the text without any effect to the graphic patterns.

●The Double Density High Resolution Graphics. Yes, you can plot 560 dots in one line. You have only 280 dots in one line on ordinary Apple-II. (Software is required)

●One of the most advanced character display. Besides Normal and Inverse characters which are already built-in you will have the choice of Half-Intensity and Hilighted characters. And more, you may Over-Write or Over-Type any character to other character if you want to do so!
(Software is Included)

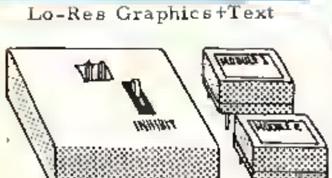
NORMAL
HALF INTENSITY

Half Intensity

S & W

Over Write

$60 Introductory price → $50
Offer ends Apr.,31,'81



COSMOS™ WORLD WIDE COMPUTER SUPER SHOP

Dealers inquiries invited.
For more information call or write to ASTAR INTERNATIONAL CO,
5676 FRANCIS AVE., CHINO, CA 91710 Phone 714-627-9887

Apple-II is a registered trademark of Apple Computer Inc.

MICRO

Software Catalog

Name: **Real-time Clock Software**
System: KIM-1
Memory: 1K
Language: 6502 Machine Language
Hardware: CMOS Real-time Clock Chip MSM5832RS
Description: *Real-time Clock Software* for KIM-1 occupies ¾K bytes plus pointers in zero page. 70-page manual includes detailed assembler listing, machine language dump and extensive documentation. User-friendly system prompts for time-settings of year, month, day, hour, minute, and seconds in a 24-hour format. Major subroutines include Read Clock, Display Time, and Write Timesetting to Clock. All subroutines are well-documented for easy modification. Manual also documents Real-time Clock circuitry required for easy construction and interconnection to KIM-1.
Price: $10.00 for Real-time Clock Software/ Hardware Manual. $1.00 for specifications/notes.
Author: Hunter Technical Services
Available: Hunter Technical Services
P.O. Box 359
Elm Grove, WI 53122

Name: **TWGI Arima**
System: Apple II, Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: 1 disk (printer and 2nd disk recommended)
Description: Box Jenkins package does seasonal and non-seasonal models. Menu-driven package does identification, estimation, diagnosis and forecasting. Includes programs to create, correct and update data; left- and right-hand software keypads. Also program to do all BASIC transformations.
Price: $250.00 includes program, test data and results, 60-page manual
Author: Eric Weiss, Ph.D.
Available: The Winchendon Group
P.O. Box 10114
Alexandria, VA 22310

Name: **Lesson-tutorgraph™ — Shore Features and Weather**
System: Apple II, Apple II Plus with Applesoft in ROM or Language System
Memory: 48K, DOS 3.3
Language: BASIC
Hardware: Apple II or Apple II Plus, 1 disk
Description: Programmed presentation of lesson material with branching, review, full color, Hi-Res illustrations, and tests. "Weather Fronts" includes front characteristics, frontal movement, and weather characteristics of different fronts. "Shore Features" teaches about beach/shore, berm, dunes, low/high tides.
Price: $24.95 includes diskette, manual with objectives, options, suggestions for use
Author: Patrick C. Moyer and Lois B. Bennett
Available: Teach Yourself By Computer (TYC) Software™
40 Stuyvesant Manor
Geneseo, NY 14454

Name: **RPL Compiler**
System: All Commodore PET and CBM-series machines
Memory: 8K minimum
Language: No additional
Hardware: No additional
Description: *RPL* (Reverse Polish Language) is a new language designed to compete with FORTH. RPL object code runs faster and takes up less memory space than FORTH object code. RPL is also much easier to use than FORTH, although it is just as powerful.
Price: $49.95 and down includes RPL compiler on disk or cassette and detailed user's manual
Author: Tim Stryker
Available: Samurai Software
P.O. Box 2902
Pompano Beach, FL 33062

Name: **EIS General Accounting Package**
System: OS65U
Memory: 48K
Language: BASIC
Hardware: Ohio Scientific C-2 or C-3 series
Description: This package includes fully integrated accounting, payroll, and inventory systems written in standard accounting terminology and procedure. It is end-user oriented, menu-driven, and generates all necessary reports and summaries. Individual Software Catalog Entry sheets on all three portions will follow in the next three months, with your one-a-month acceptance plan.
Price: $3,500.00 total system includes program disks, data disks, and complete manual
Author: Electronic Information Systems, Inc.
Available: Electronic Information Systems, Inc.
P.O. Box 5893
Athens, GA 30604
(404) 353-2858

Name: **Rubin's Cube**
System: OSI C1P
Memory: 8K
Language: 8K BASIC in ROM
Description: Try to solve this popular cube puzzle on your computer. The program displays a large three-dimensional cube that just fits on the 24 × 24 screen and uses graphics in place of colors. The computer mixes up the cube and, through the keyboard, you can make any move that can be made with the real cube to solve the puzzle. The commands are easy to learn and only four characters long.
Price: $5.00 includes cassette and documentation
Author: Brian Zupke
Available: B.C. Software
9425 Victoria Drive
Upper Marlboro, MD 20772

Name: **Accounts Receivable**
System: Apple II
Memory: 48K
Language: Applesoft or Language System
Hardware: Dual 5" drives, any 130-column printer
Description: A quality program, structured around the Osborne Accounts Receivable software, with several added enhancements. Can be used alone or integrated with *General Ledger*. Features open invoicing, credit and debit memos, full or partial payments,

progress billing, invoice aging and printing of statements. System is available on DOS 3.2, DOS 3.3, or 8" drives, if additional capacities are required.

Price: $249.00
Author: David McFarling
Available: Small Business Computer Systems
4140 Greenwood
Lincoln, NE 68504
(402) 467-1878

Name: **Type**
System: SDOS or SDOS/MT
Memory: 48K minimum
Hardware: 6800/6809 CPU with CRT, disk and printer
Description: *Type* is a document-formatting program, used in word processing or document production. Commands embedded in raw text files processed by *Type* control the formatting of that text on the output device. Output formatting includes full justification, page width and depth, page numbering, centering, spacing, titles and table of contents generation. *Type* is used in conjunction with the SD Screen Editor for easy data entry.
Price: $140.00 includes program, 100-page manual
Author: AMS
Available: Software Dynamics, exclusively
2111 W. Crescent, Su. G.
Anaheim, CA 92801
(714) 635-4760

Name: **Fast Facts**
System: Apple II, Apple II Plus
Memory: 48K
Language: Applesoft
Hardware: Disk 3.2 or 3.3, line printer desirable
Description: This selection of programs was created and designed by a certified financial planner for quick analysis of the personal investment planning needs of his clients. It was professionally programmed for efficient and accurate operation. *Fast Facts* operates easily with single key program selection and printing commands. In many cases the entire planning sequence is completed in less than 60 seconds. Specific program objects are divided into six systematic program fields. They are: planning for retirement; college financing; diversifying investments; results of inflation in devaluing earnings; costs of borrowing money and loan balance at any point in time; investment calculations for compounding and future values.
Copies: Version 1.1 just released
Price: $95.00 includes disk and instructions
Author: Monte C. Fremouw
Available: Richard Lorance CPF
c/o Richard Lorance and Associates, Ltd.
3336 N. 32nd Street, Suite 102
Phoenix, AZ 85018

Name: **Loan Pack**
System: OSI Challenger (C2 or C3)
Memory: 48K
Language: BASIC (under either OS65-U or 65-D)
Hardware: Disk drive, CRT, printer optional
Description: *Loan Pack* is a loan analysis package. It computes either the interest rate of a loan, the principal amount of a loan, the amount of the loan payment, or the number of payments required to pay a loan, when any three of the other variables are known. It also calculates the present unpaid principal of a loan which is partially paid; and calculates and displays (or prints) a loan amortization schedule showing allocation of payments between interest charges and principal loan payments, and totals amounts of interest paid in each calendar year of the loan. Self documenting.
Price: $5.00 for listing
$12.00 for 8" disk (specify 65-U or 65-D)
Author: Bob Sullivan
Available: Professional Computers, Incorporated
10885 Washington Blvd.
Culver City, CA 90230
(213) 836-5005

Name: **VIC Software**
System: VIC 20
Memory: 3K
Language: BASIC and Machine Language
Description: Arcade games are now available for the VIC. *Cattle Roundup* has you round up eight cattle in a maze. *Artillery* lets you exchange fire over a mountain. *Target* is based on the arcade game *Missile Command*.
Price: $9.95 each includes cassette and program
Author: Cliff Dudzik
Available: Computermat Software
P.O. Box 1664
2984 Daytona
Lake Havasu, AZ 86403

Name: **World and State Capitals**
System: Apple II, Apple II Plus
Memory: 36K with DOS 3.3 II or 3.2 II and FP installed
Language: Applesoft BASIC
Hardware: DOS 3.2 II or DOS 3.3 II with controller card, printer option
Description: Consists of a 100-world nation/capital and a 50-U.S. state/capital test. Both tests display a low-resolution graphic which animates dripping square-dots to fill the shape of the capital building. Right, wrong, total score points plus a bonus feature triggers this animation. The program features a simple single-key stroke answering system (except spelling test). This eliminates the standard use of the return key. The program was designed to eliminate teacher's supervision and encourage student's spelling accuracy, knowledge of the U.S. and the world. The score results are retrievable for teacher's use only (by date and name, stored score order sequence). In addition to above, the teacher can produce his or her own test, using the same features and format for up to 100 questions.
Price: $25.00 for both programs with test report and test editor/report on a single diskette
Author: American Avicultural Art & Science Inc.
Available: American Avicultural Art & Science, Inc.
3268 Watson Road
St. Louis, MO 63139

Name: **Household Finance**
System: Commodore VIC-20
Memory: 3.5K User RAM (unexpanded VIC)
Language: BASIC
Hardware: Commodore VIC-20 with cassette unit
Description: Four programs to record household expenses and income in 16 categories. Provides monthly and yearly totals in tabular and graphic form. Handles budgeting, and sums tax-deductibles.
Price: $34.95 includes two cassettes containing four programs, instruction booklet.
Author: John C. Doering, and Paul Zuzelo
Available: Creative Software
201 San Antonio Circle #270
Mountain View, CA 94040
(415) 948-9595

Name: **Geomap**
System: Apple II or III (adaptable to other systems)
Memory: 32K
Language: Machine
Hardware: One disk, printer and/or plotter
Description: This is a contour mapping package with the following features: menu driven; easy to use; choice of several map styles; adapts to user's printer width; maps are made in strips the width of printer or plotter paper (no limit to strip length or number of strips in single map); accepts either gridded or irregular data; large regions are subdivided into small parcels which can be run individually and randomly to form a contiguous, integrated whole (which minimizes RAM requirements and permits excessively long runs to be broken down into several small runs — an important feature for micros); modular design; choice of gridding algorithms; training available.
Price: $2,000.00 (Manual alone $5.00. Refundable with purchase.)
Author: Mason Christner
Available: Geosystems, Inc.
802 E. Grand River
Williamston, MI 48895

Name: **Egbert RTTY Program**
System: Apple II, Apple II Plus
Memory: 48K
Language: Applesoft (ROM) and Machine Language
Hardware: Apple disk with DOS 3.2 or 3.3
Description: Transmit and receive RTTY without any expensive interface hardware. The Apple cassette ports connect directly to the transmitter/receiver — no additional hardware required! The Apple generates and decodes the RTTY tones. Program capabilities include 60, 67, 75, and 100 WPM Baudot and 110 Baud ASCII, unique receiver tuning using the Hi-Res graphics, type-ahead-buffer, break without dumping the type-ahead-buffer, canned messages, save received text/pictures to disk, automatic C.W.I.D., game port-driven push-to-talk, plus more.
Price: $39.95 (California residents add 6% tax) includes program disk and instruction manual
Author: G.W. Egbert
Available: W.H. Nail Co.
275 Lodgeview Dr.
Oroville, CA 95965

Name: **6809 Pascal Compiler**
System: 6809 FLEX™ or 6809 UniFLEX™
Memory: 56K minimum
Language: Pascal
Hardware: Any that supports standard 6809 FLEX or UniFLEX
Description: Native-code Pascal compiler generates assembly language source which is assembled into true 6809 object code. This results in faster program execution speeds than common "P-code" interpreters. Supports nearly all of the Jensen & Wirth Pascal specifications plus additional features. Includes both integer and floating point math with up to 16.8 digits of accuracy.
Price: $200.00 for FLEX version; $300.00 for UniFLEX version. Includes user's manual, *Pascal User Manual & Report* by Jensen & Wirth, and object code on diskette.
Available: Technical Systems Consultants, Inc.
P.O. Box 2570
West Lafayette, IN 47906
(317) 463-2502
Telex: 276143

Name: **Presidential Campaigns**
System: Ohio Scientific
Memory: 8K
Language: BASIC
Hardware: C2-4P, C2-8P, C-4P, C-8P (Polled Keyboard)
Description: The program gives the user the opportunity to vote for every U.S. President from 1788 to 1980, and advises if his or her candidate won or lost, giving the name of the winner and his Vice-president.
Price: $9.95 includes cassette
Author: John and Mary Neally
Available: SouDustrial Electronics, Incorporated
4066 Polaris Avenue
Joshua Tree, CA 92252
(714) 366-9572

Name: **Lightning-Bolt**
System: OSI C4PMF
Memory: 24K
Language: BASIC
Hardware: One disk drive
Description: The finest D&D adventure for the OSI computer yet! This adventure is so comprehensive that it takes up the entire disk. You must cross mountains, forests, great plains, and oceans to seek your fame and fortune! This game has full color graphics as well.
Price: $29.95 includes comprehensive 8-page manual
Author: Steve Brown
Available: Interesting Software
15217 Campillos Rd
La Mirada, CA 90638

Name: **Jabbertalky**
System: Apple II, TRS-80
Memory: 16K TRS-80 cassette, 32K TRS-80 disk, 48K Apple disk
Language: Applesoft or TRSDOS
Hardware: Apple II, TRS-80 (model I, level II and model III), cassette or disk drive
Description: A programmable word game for one or more players, *Jabbertalky* includes two game features and a utility program. "Alphagrammar," an anagram game, challenges players to unscramble entire grammatically correct sentences. In "Cryptogrammar," a code breaking game, the player must decode sentences in which each letter of the alphabet is substituted for by another. The utility program lets players create their own sentences. *Jabbertalky* has eight skill levels and is for ages seven through adult.
Price: $29.95 includes game box, rule book, loading instructions and disk or cassette
Available: Automated Simulations, Inc.
P.O. Box 4247
Mountain View, CA 94040

Name: **Painter Power**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft in ROM
Hardware: Disk II
Description: Anyone can create computer art. Using the beginner or advanced mode, children and adults can create original art designs then use them, or any other saved screen, to prepare slide shows and art demonstrations.
Price: $39.95
Author: Eric Podietz
Available: Micro Lab
2310 Skokie Valley Rd.
Highland Park, IL 60035

MICRO

Hardware Catalog

Name: **Execom 80**
System: CBM/PET 2001/3000/4000 Series
Memory: Additional 1K
Language: Operating System
Hardware: 1 4″×5″ and 1 2″×2″ circuit board with an optional 2″×2″ circuit board
Description: 80-column expansion that allows the user to switch between 80-column and 40-column displays, from program control, or directly from the keyboard. Requires some circuit modification to CBM/PET circuit board.
Price: $275.00 includes all necessary hardware and ROMS, demo diskette (4040 format)
Available: Execom Corp.
1901 Polaris Ave.
Racine, WI 53404

Name: **HIPLOT Plotters**
System: TRS-80, 1/2/3, Atari, Apple, PET
Memory: 16 K
Language: User's Choice
Hardware: High-resolution pen plotter and interface card (if needed)
Description: *HIPLOT Plotters* is a very high quality alternative. Multiple pen colors is an add-on option. Resolution is typically 200 points/inch. Several different models are available in different sizes and options. A high-level plotting language drives the Z-80-based hardware. Software is available.
Price: $1025 - $2000
Available: Houston Instruments
One Houston Square
Austin, TX 78750

Name: **Analog Peripheral**
Hardware: Any system with RS-232 port
Description: Self-contained 8-bit analog-to-digital converter. RS-232C output line is switch-selectable from 110 to 9600 baud. For faster data transfer, there is also a 26-pin parallel output. Plug-in transducers eliminate need for breadboarding transducer circuits. Four input channels permit logging of several variables at once. Fast conversion speed of 100 microseconds. One BASIC instruction begins data logging.
Price: $449.00
Available: Cambridge Development Laboratory
36 Pleasant Street
Watertown, MA 02172

Name: **Sabrina**
System: SCS-10
Memory: 10 to 120 megabytes
Language: Apple 3.3 DOS, CP/M, Pascal 1.1, TRS DOS
Hardware: Winchester disk subsystem
Description: *Sabrina* is the SCS-10 8″ Winchester hard disk storage system that will interface to over 9 different major microcomputers including Apple II, TRS-80, S-100, Multibus, and the IBM personal computer.
Price: Starting at $4,995.00 includes everything required to run the hard disk to the host
Available: Santa Clara Systems
560 Division St.
Campbell, CA 95008
(408) 374-6974

Name: **Sun-Flex Touch Pen System**
Description: Microprocessor-based, stylus-operated, graphic-capable interface, which enables a CRT operator to bypass the keyboard and communicate directly with the CPU. Available in sizes up to 25″.
Price: $250.00 - $1,000.00 includes transparent screen panel, stylus, microprocessor
Available: Sun-Flex Company, Inc.
20 Pimentel Court
Novato, CA 94947

Name: **The Grappler**
System: Apple II, Apple II Plus
Language: All
Hardware: Parallel interface board for Apple
Description: *The Grappler* interface is the first universal parallel interface card to provide sophisticated on-board firmware for Apple high-resolution graphics. No longer does the user need to load clumsy software routines to dump screen graphics — it's all in the chip. Actually, it's our EPROM, and there are versions to accommodate numerous printers. *The Grappler* accepts 18 simple software commands accessible through the keyboard or user program, making it the most intelligent Apple interface available. It is also Pascal- and CP/M-compatible.
Price: Includes 5 ft. cable and manual
Available: Orange Micro
3150 E. La Palma, Suite 1
Anaheim, CA 92806
(800) 854-8275

Waybern
13911 Enterprise Dr.
Garden Grove, CA 92643
(714) 554-4520

CompuCable
2081 Business Center Dr.
Suite 180
Irvine, CA 92715
(714) 635-7330

Name: **HTS General Expansion Boards**
System: KIM-1 and OSI C1P
Memory: 2K bytes 2114L Static RAM, 4K bytes 2716 PROM
Hardware: 32 lines I/O port
Description: Low-cost general expansion boards for the KIM-1 or OSI C1P. Occupies 8K block of memory, location is switch selectable on any 8K boundary. Sixty-plus page manual describes two board designs, GEBI uses 2K of 2114L RAM while GEBII uses two 2K×8 RAM chips. GEBII ports overlap 64 bytes at upper boundary of RAM. Both boards provide 32 port lines using 6532 ports with instructions included for substituting 6522 VIA. Manual provides wiring diagrams, wire lists, parts lists and instructions for wire-wrap or point-to-point construction on Vector boards. Buffered bus is similar to KIM-1 structure. I/O connections made *via* DIP sockets. Connection details for KIM-1 and OSI C1P extensively detailed in manual. Can also be used for AIM 65 and SYM.
Price: $10.00
Available: Hunter Technical Services
P.O. Box 359
Elm Grove, WI 53122

MICRO

Next Month in MICRO

FORTH Feature

- **LIFE in FORTH and BASIC —** FORTH and BASIC versions of the educational game LIFE are presented, with a detailed description and comparison.
- **Using FORTH with the 6502 —** A complete extension, a debugging or decompiling tool, and an application utility all built into a flexible FORTH enhancement, with enough room for other similar expansions and applications.
- **Stepper Motor Control — A FORTH Approach —** Stepper motors translate digital commands to motion, bridging the gap between computers and robots. A flexible command language, written in FORTH, translates natural, English-like commands to precisely controlled movement.
- **FORTH: A Viable Alternative —** An introduction to FORTH, from FORTH, Inc.

Other Features

Inspector (for the TRS-80C Color Computer)
AIM Assembler Listings
List-Corrupted SYM-bols
Atari Countdown Timer
Applesoft Input Anything
Fast Joystick Input for C1P
Credit Box Creator for the Apple

Columns

Pet VFT
From Here to Atari
New! The Single Life (for single board users)

Plus Our Regular Departments

Software Catalog
Hardware Catalog
6502 Bibliography
...and more!

HUNTINGTON COMPUTING

ONE OF THE WORLD'S LARGEST INVENTORIES

VisiCalc List $200.00 .... now **$149.00**

VisiFile List $250.00 .... now **$199.00**

Horriblescope (Disk) List $15.00 .... now **$3.99**

MISCELLANEOUS

Escape from Arcturus	$29.95 now	$35.39
Basic Mailer	$69 95 now	$59.49
Memory Management II	$49 95 now	$43.89
Castle Wolfenstein	$29 95 now	$25.39
Upper Reaches of Apshai	$19 95 now	$14.89
Bridge Tutor	$39 95 now	$35.19
PASCAL Animation Tools		$65.99
Hand Holding BASIC		$84.99
UT-100 Emulator		$65.99
Space Quark	$29.95 now	$25.29
Beneath Apple DOS (book)	$19.95 now	$16.89
Birth of the Phoenix	$14 95 now	$12.49
Geblins	$27 50 now	$33.39
Painter Power	$39 95 now	$33.89
U S Constitution	$29 95 now	$25.39
Merger	$49 95 now	$42.49
Super Stellar Trek	$39.95 now	$33.89
LISA	$79.95 now	$67.89
V-Plot (Yuccipa)	$29 95 now	$25.39
V-Stat	$29 95 now	$35.39
V-Print	$29 95 now	$25.39
Brain Surgeon	$49.95 now	$43.49
Info Master	$150 00 now	$137.49
Waterloo II	$49 95 now	$43.39
Fantysland 2U41	$59 95 now	$50.99
Torpedo Terror	$24 95 now	$21.19
Speedstar	$139 95 now	$114.69
Kaves of Karkhan	$49 95 now	$42.39
Dos Boss	$24 00 now	$20.39
Memorex Disks		10 for $34.99
Verbatim Datalife-plain w/hubs		10 for $27.99
Dysans		10 for $49.99
Flipsort Box		$39.80
Scotch Disk Cleaner	$29 95 now	$24.99
Atari*/TRS-80*/Pet*		Write for Information

GAMES

Hi-Res Soccer	$29 95 now	$25.39
Apple*-Oids	$29 95 now	$35.39
Wurst of Huntington Computing		$19.99
Gobbler	$24 95 now	$21.19
Ultima	$39 95 now	$33.89
Autobahn	$29 95 now	$25.39
Battle Cruiser Action	$44 95 now	$38.89
Gorgon	$39 95 now	$33.89
Super Stellar Trek	$39 95 now	$33.89
Hellfire Warrior	$39.95 now	$33.99
Gamma Goblins	$29.95 now	$35.39
Mission Asteroid	$19 95 now	$17.99
Wizardry	$49 95 now	$43.49
Star Mines	$29.95 now	$35.39
Warp Factor	$39.95 now	$33.99

WURST OF HUNTINGTON

Read Softalk's review on page 87 of the October issue

$19.99

NEW PROGRAMS

Sneakers	$29 25 now	$25.39
Cops & Robbers	$34.95 now	$39.49
Pot-O-Gold I	$39 95 now	$33.89
Pot-O-Gold II	$39 95 now	$33.89
Ring of Saturn	$39 95 now	$33.49
Streets of the City & Trucker	$24 95 now	$31.19
President Elect	$39.95 now	$33.89
Dis-o-Doc	$72 39 now	$41.49
Med Fly Mania	$29 95 now	$24.39
Hungry Boy	$24.95 now	$21.19
Anova	$74.95 now	$43.79
Solitaire	$29 95 now	$35.39
The Source	$100.00 now	$89.99
Race for Midnight	$29.95 now	$35.39
Hi-Res Secrets	$125 00 now	$104.19
Banners I	$34 95 now	$29.69
Complete Graphics (Tablet)	$119 95 now	$101.89
Special Effects (Tablet)	$69 95 now	$60.79
Inkwell Apple* III	$150 00 now	$130.49
Snack Attack	$29 95 now	$35.39
V-Plot	$29 95 now	$25.29
Disasem/65	$29 95 now	$25.39
The Inspector (Disk or Prom)	$49.95 now	$43.39
The Book	$19 95 now	$11.97
Don't (Book)	$8 95 now	$8.05
Using 6502 Assembly Language	$19 95 now	$16.49
Expandaport	$59 95 now	$50.89
Firebird	$29.95 now	$35.39
Thief	$29 95 now	$25.39
Threshold	$23 97 now	$38.38
Target Planner	$195 00 now	$145.49
Paddle-Adapple	$29 95 now	$36.89
So. Cal Research	$14 95 now	$13.45
Word Handler	$250 00 now	$213.49

GAMES

Microsoft Adventure	$29.95 now	$35.39
Wizard and the Princess	$32 95 now	$28.99
Flight Simulator	$34 95 now	$29.69
Odyssey	$29 95 now	$35.39
Sargon II	$34 95 now	$29.69
Space Eggs	$29 95 now	$35.39
Hi-Res Cribbage	$24 95 now	$21.19
Lords of Karma (cass)	$20 00 now	$16.99
Oh Shoot	now	$19.99
ABM	$24 95 now	$21.19
Computer Conflict	$39.95 now	$25.99
Computer Air Combat	$59 95 now	$53.99
Temple of Apshai	$39.95 now	$23.89
Zork	$39 95 now	$33.99
All Nibble Software		15% off list
Robot Wars	$39 95 now	$33.99
Cranston Manor	$34 95 now	$39.49
Dragon's Eye	$24 95 now	$21.19
Computer Acquire	$20.00 now	$16.99
Twala's Last Redoubt	$29 95 now	$25.29
Alien Rain	$24 95 now	$31.19
Alien Typhoon	$24 95 now	$21.19
Raster Blaster	$29 95 now	$35.39
3-D Skiing	$24 95 now	$21.19
Creature Venture	$24 95 now	$21.19
Galaxy Space War I	$39.95 now	$33.99
Meteoroids in Space	$19.95 now	$14.99
Dragon Fire	$49 95 now	$43.49
Pool 1 5	$34 95 now	$39.69

NEC Green Screen Monitor List $285.00 .... now **$199.00**

Zenith Green Screen Monitor List $159.00 .... now **$109.00**

Ulysses List $34.95 .... now **$29.69**

Speed/asm List $39.95 .... now **$33.89**

Lisa 2.5 List $79.95 .... now **$67.89**

Dinosaurs List $15.00 .... now **$12.69**

BMP Joystick

Here's a joytsick you won't want to be without. It's the only joystick we know about that allows you to change the axis with a flip of a switch. Only the highest quality parts are used.

A real bargain at

$49.95

Apple®

ORDER BY PHONE
800-344-4111 (outside Calif.)
800-692-4143 (inside Calif.)
209-992-5411 (foreign or local)

HUNTINGTON COMPUTING
Post Office Box 787
Corcoran, California 93212

VISA

master charge

We take MasterCard, VISA or American Express (Include card # and expiration date). California residents add 6% tax. Include $2.00 for postage. Foreign and hardware extra. Send for free catalog. Prices subject to change.